03-713 Group C
Yuting Xiao, Xiaoqi Fang, Xinling Li, Zhenyu Yang

Final Report

1. Introduction to our pipeline

Open reading frame(ORF) is a short segment of DNA that can be translated and contains start(AUG) and stop codon(UAA,UAG,UGA). It's very important for gene annotation. In this project, we mainly focus on finding ORFs in the whole genome sequence.

In general, there are two approaches for genome annotation: de novo approach and Homology-based method. In our project, we firstly managed to perform homologous annotation for mouse and human gene based on the De novo annotation method developed using a package called RiboCode[1]. Then, we tried to combine two pipelines called ORFM and ORF-finder to create a pipeline based on Aho-Corasick algorithm, and improve its accuracy by embracing several limitations for ORF length and codon stringency. .

RiboCode is a software that utilizing ribosome profiling data to identify the ORFs of a whole genome for one particular specie. It provides real-time snapshots of translation (translatome) across the whole transcriptome[1]. De novo annotation is a method of a transcriptome assembly. It is often the preferred method to studying non-model organisms, since it is cheaper and easier than building a genome, and reference-based methods are not possible without an existing genome[2]. The RiboCode assembly is based on De novo annotation method.

The purpose of this project is to create a software pipeline that performs gene annotation by integrating available packages. Our pipeline contains two parts. The first part focus on incorporating RiboCode and combining with BLAST search, and the second part is designed as another approach for predicting ORFs. The first part of the program can be run in a single python program named "pipeline.py", and all the scripts needed for running part 2 is integrated in the file named "pipeline_part2.py".The final result returned by our pipeline is chosen and considered with high quality result. Softwares used in our pipeline including Bowtie[3], STAR, RiboCode, ORF-Finder and BLASTP. We first perform RNA transcriptome data preprocessing by remove 3'-adaptor from transcriptome data and low quality reads with low quality scores. Then, Bowtie compress genomic data using Burrows-Wheelers and identifies and discards sequencing reads originating from rRNA by aligning it to rRNA sequences of human and mouse.Next, STAR maps the remaining reads to the genome and spliced transcripts. In the end, BlastP aligns the protein sequence of ORFs found in RiboCode to protein database to see which one is protein coding sequence.

We developed our own package by integrating other existing packages to predict the ORF according to the raw data set. Also, we compared our package with other teams' pipelines. Detailed comparisons are included in the discussion section.

1.2. Required Inputs:

Required input files include:

1. **Ribosome profiling data in fastq format**: sequence of ribosome-protected mRNA fragments [4]
2. **Whole genome sequencing data in fasta format**
3. **rRNA data in fasta format, file name should start with "rRNA"**
4. **GTF of the second file:** gene transfer format which is a file that holds information about gene structure [5]

2. Part I Results:

Our group assembled the RiboCode according the menu. Both mouse data and human data is prepared to be the input of the RiboCode. The result is shown as below:

2.2. Human:

1) The category of ORFs are 88.4% annotated, 3.6% uORF, 1.1% dORF, 3.5% overlap, 0.5 % novel_PCG, 2.8% novel_NonPCGs(Fig1).

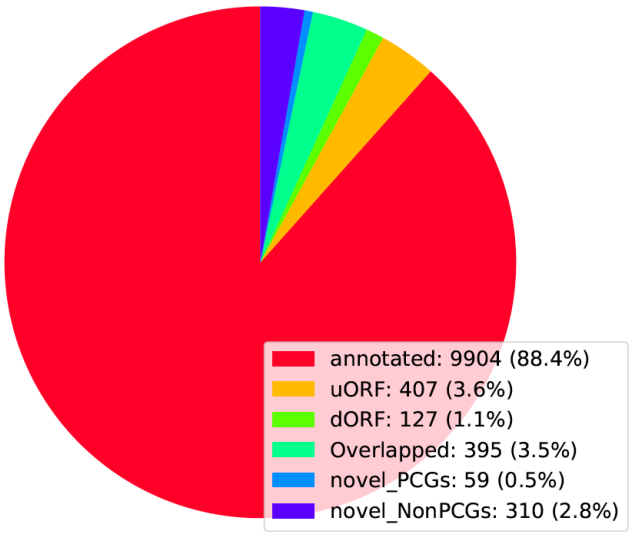


Figure 1. Category of ORFs

2) Meta Plots(Figure 2). The figure below is one of the figures that's generated by Ribocode. Other figures can be found in supplement. They are histograms of distance from 5' end of ribosome protected RNA fragments to the annotated start or stop codon. The length of ORF is 25 nucleotide long. X axis is the distance from each read to 5' - start codons or 5' - end codons. 0 site represents the first nucleotide of the start or stop codon. Y axis is the number of alignments .Red represents frame 0, blue represents frame +1, and green represents frame +2. Frame 0 has highest number of alignments. As can be seen from the top figure of figure 2, the distance between most 25-nucleotide long RPF and start codon is about 9 nucleotides. For the bottom figure of figure 2, the distance between most 25-nucleotide long RPF and stop codon is about 20 nucleotides. Both figures show frame 0 has highest alignments.

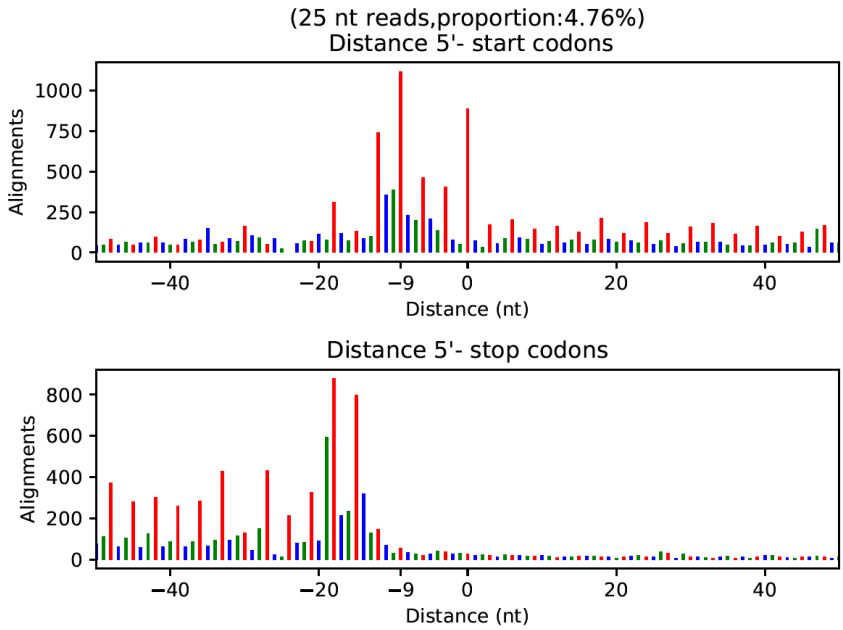


Figure 2: Metagene analysis for RPF length selection and P-site identification.

3) ORF result. The result of the RiboCode is Abundant. The file contains: ORF_ID, ORF_type(novel and annotated), transcript_id, transcript_type, gene_id, gene_name, gene_type, chromosome, strand, ORF_length ,ORF_tstart, ORF_tstop, ORF_gstart, ORF_gstop, annotated_tstart, annotated_tstop, annotated_gstart, annotated_gstop, Psites_sum_frame0, Psites_sum_frame1, Psites_sum_frame2, Psites_coverage_frame0, Psites_coverage_frame1, Psites_coverage_frame2, Psites_frame0_RPKM, pval_frame0_vs_frame1, pval_frame0_vs_frame2, pval_combined, and AAseq. The smaller the p-value, the more significant the ORF is. We used the amino acid sequence to align against the protein database in Blast.

4) ORF counts. All the counts are 0. We wrote to the author but received no response.

2.3. Mouse:

1) The category of ORFs are 96.7% annotated, 0.9% uORF, 0.4% dORF, 1.2% overlap, 0.1 % novel_PCG, 0.6% novel_NonPCGs (Figure 3)

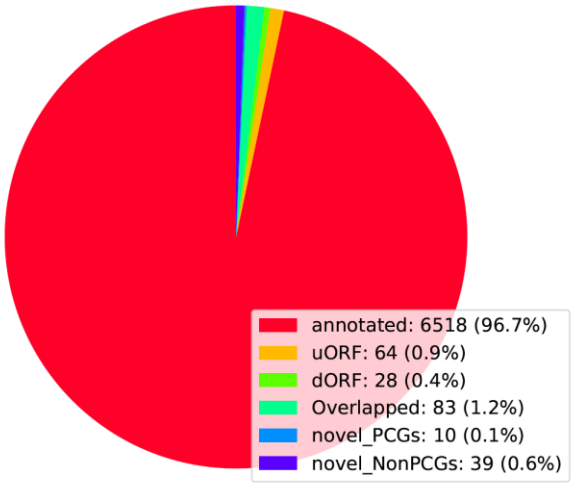


Figure 3. Category of ORFs

2) Meta Plot (Figure 4):The figure below is one of the figures that's generated by Ribocode. Other figures can be found in supplement. The length of ORF is 25 nucleotide long. X axis is the distance from each read to 5' - start codons or 5' - end codons. 0 site represents the first nucleotide of the start or stop codon.Y axis is the number of alignments .Red represents frame 0, blue represents frame +1, and green represents frame +2. Frame 0 has highest number of alignments. As can be seen from the top figure of figure 4, the distance between most 25-nucleotide long RPF and start codon is about 10 nucleotides. For the bottom figure of figure 4, the distance between most 25-nucleotide long RPF and stop codon is about 25 nucleotides. Both figures show frame 0 has highest alignments.

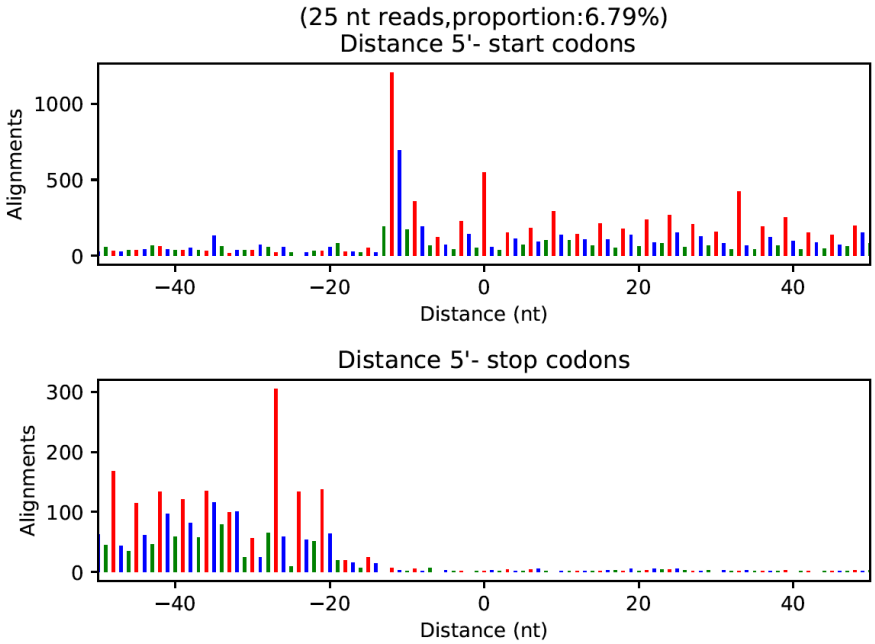


Figure 4: sample result

3) 4) The categories in ORF result file is the same. ORF counts are also zero.

2.4. BLAST

After obtaining RiboCode result, our pipeline further examine the novel ORFs by running BLAST searches. BLAST finds regions of similarity between biological sequences. The program compares protein sequences to sequence databases and calculates the statistical significance, search results can be used as validation of RiboCode results.

2.4.1 The following command is used to extract all novel sequences obtained from RiboCode. Run the command line on your terminal, the output file "tmp.txt" contains only rows in the original result that have been labeled as "novel" ORFs:

```
awk '{if ($2 == "novel") {print}}' RiboCode_ORFs_result.txt > tmp.txt
```

2.4.2 The following command is used to extract the protein sequences for each row in the input file "tmp.txt". Run this command on your terminal, the output file "sequences.txt" contains protein sequences for all "novel" ORFs:

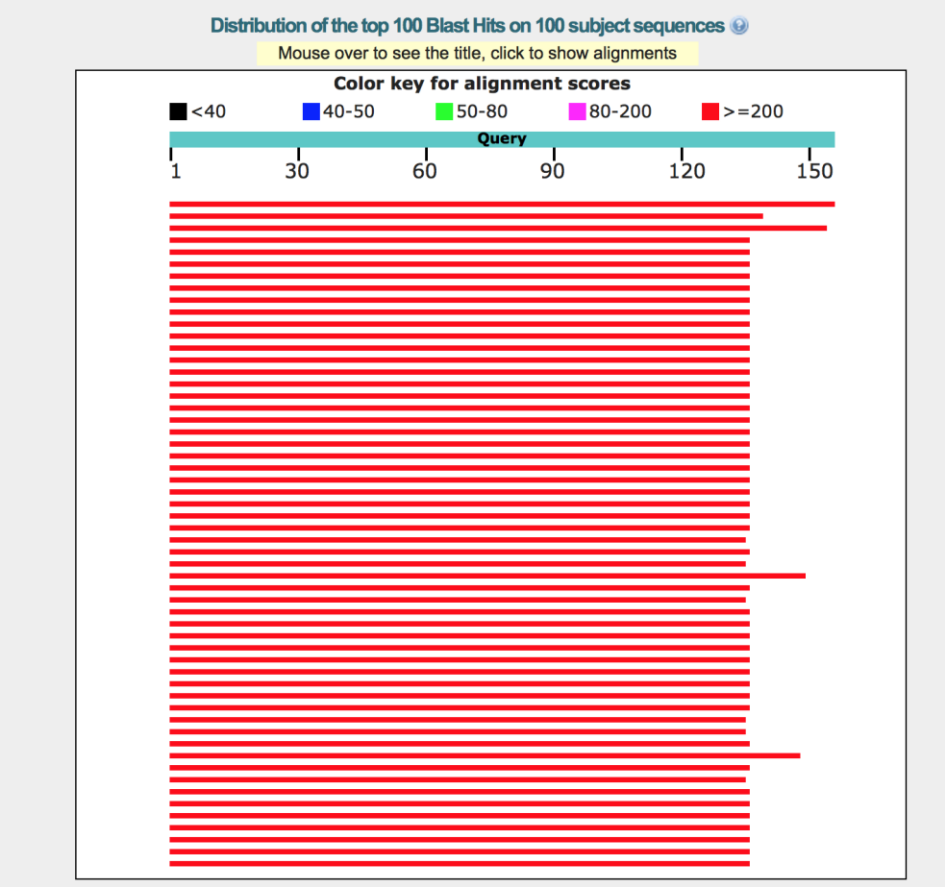
```
awk '{print $29}' tmp.txt > sequences.txt
```

2.4.3 Open a BLAST website: https://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastp&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome, and copy the contents in sequences.txt to this web site. You will get the results. It usually takes days to run this step. Since this step takes too long, our group only selected several sequences and run BLAST. An sample result is shown below.

2.4.4 Sample result for query a single amino acid sequence against the protein database sequence(MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAA RELLTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLAKSI HHARVLIRQRHIRYHLGWAPESSSTCPSDGCPH)

1) Summary:

As can be seen from the figure below, the number of hit is greater than 200, which suggest that the sequence we queried and the one in database has high similarity.



2) Descriptions:

As can be seen below, it's a real ORF, because it matches 100% to the sequence in the protein database, which suggests that the ORF that we find can be translated into a protein.

Descriptions

Sequences producing significant alignments:

Select: All None Selected:0

Alignments Download GenPept Graphics Distance tree of results Multiple alignment

Description	Max score	Total score	Query cover	E value	Ident	Accession
ribosomal protein S9, isoform CRA_e [Homo sapiens]	318	318	100%	8e-110	100%	EAW72214.1
RPS9 isoform 9 [Pongo abelii]	276	276	89%	8e-94	100%	PNJ84497.1
hypothetical protein Celaphus_00004894 [Cervus elaphus hippelaphus]	281	281	98%	3e-93	93%	OWK16473.1
40S ribosomal protein S9 isoform X3 [Physeter catodon]	270	270	87%	2e-91	100%	XP_023988187.1
PREDICTED: 40S ribosomal protein S9 [Jaculus jaculus]	270	270	87%	4e-91	100%	XP_012807417.1
PREDICTED: 40S ribosomal protein S9 [Serinus canaria]	270	270	87%	6e-91	100%	XP_009098805.1
PREDICTED: 40S ribosomal protein S9 [Hipposideros armiger]	270	270	87%	9e-91	100%	XP_019481278.1
40S ribosomal protein S9 [Phascolarctos cinereus]	271	271	87%	1e-90	100%	XP_020859361.1
hypothetical protein AB205_0048320 [Rana catesbeiana]	268	268	87%	1e-90	99%	PIO23518.1

3) Alignments:

Download GenPept Graphics ▼ Next ▲ Previous Descriptions

ribosomal protein S9, isoform CRA_e [Homo sapiens]

Sequence ID: EAW72214.1 Length: 156 Number of Matches: 1

See 1 more title(s)

Range 1: 1 to 156 GenPept Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps
318 bits(814)	8e-110	Compositional matrix adjust.	156/156(100%)	156/156(100%)	0/156(0%)

```
Query  1    MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL  60
            MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL
Sbjct  1    MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL

Query  61   LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA  120
            LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA
Sbjct  61   LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA

Query  121  KSIHHARVLIRQRHIRYHLGWAPESSSTCPSDGCPH  156
            KSIHHARVLIRQRHIRYHLGWAPESSSTCPSDGCPH
Sbjct  121  KSIHHARVLIRQRHIRYHLGWAPESSSTCPSDGCPH  156
```

Related Information

Gene - associated gene details

Identical Proteins - Identical proteins to EAW72214.1

Download GenPept Graphics ▼ Next ▲ Previous Descriptions

RPS9 isoform 9 [Pongo abelii]

Sequence ID: PNJ84497.1 Length: 139 Number of Matches: 1

Range 1: 1 to 139 GenPept Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps
276 bits(707)	8e-94	Compositional matrix adjust.	139/139(100%)	139/139(100%)	0/139(0%)

```
Query  1    MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL  60
            MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL
Sbjct  1    MPVARSWVCRKTYVTPRRPFEKSRLDQELKLIGEYGLRNKREVWRVKFTLAKIRKAAREL

Query  61   LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA  120
            LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA
Sbjct  61   LTLDEKDPRRLFEGNALLRRLVRIGVLDEGKMKLDYILGLKIEDFLERRLQTQVFKLGLA

Query  121  KSIHHARVLIRQRHIRYHL  139
            KSIHHARVLIRQRHIRYHL
Sbjct  121  KSIHHARVLIRQRHIRYHL  139
```

Related Information

3. Part II: Improve the performance of Aho–Corasick algorithm by RiboCode-annotated transcripts

3.1 The structure of reading frames on DNA

DNA is translated in group of codons, and each codon has 3 nucleotides. If we choose to start at -1, 0, +1 position of a start codon, there are 3 different reading frames to interpret the genome sequence. Moreover, each DNA molecule has two strands complementary to each other and form a stable double helix structure, and the open reading frame can be located on both strands. Therefore, it is necessary to consider the corresponding 3 reading frame in the opposite direction.

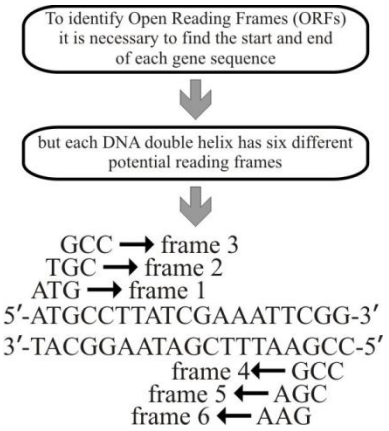


Figure 5. 6 reading frames of double-stranded DNA[1]

3.2. Aho-Corasick Algorithm and Its application in Genome Mapping

Aho-Corasick algorithm is a string-searching algorithm designed by Alfred V. Aho and Margaret J. Corasick. It can locate specific strings in a finite set within an input text, and the most appealing feature of this algorithm is it can locate all specified strings simultaneously. This attractive feature is achieved by the data structure named keyword tree, or prefix tree.

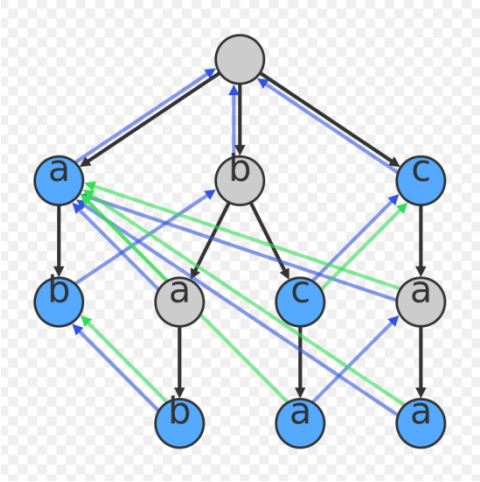
3.2.1 Key-word Tree

A keyword tree is an effective implementation of strings stored in a dictionary. To understand what Aho-Corasick algorithm do when searching and mapping strings, intuitively speaking, there is a set of words and a long text, and the mission is to find all the locations of the words within the text.

However, there are several limitations for completing this mission: first, you can access only 1 character of the text each time; second, when you go over one character, you can never look back again; third, you can only access the whole text once to figure out all the locations of the words from the set.

For example, if there is a word “ORF” in the set you want to locate in the text, when you encounter an “O”, you should remember this site since it’s a possible prefix of “ORF” and is likely followed by “R”. It is reasonable to remember all possible prefix for words in the set, and determine whether to keep this memory or not once go through the word. After finish traversing through the text, all the words we are interested in the text will be located, if exists.

As noted in the graph below, we have a set of words P = {a, ab, bab, bc, bca, c, caa}, and the keyword tree constructed by AC algorithm for this set will be:



Dictionary {a, ab, bab, bc, bca, c, caa}

Path	In dictionary	Suffix link	Dict suffix link
()	–		
(a)	+	()	
(ab)	+	(b)	
(b)	–	()	
(ba)	–	(a)	(a)
(bab)	+	(ab)	(ab)
(bc)	+	(c)	(c)
(bca)	+	(ca)	(a)
(c)	+	()	
(ca)	–	(a)	(a)
(caa)	+	(a)	(a)

Figure 6. Keyword tree generated by AC algorithm[2]

In this table, each row represents a node in the trie, and the path goes through the column indicates the sequence of characters from the root to leave.

For every string in the dictionary, it has several nodes to store all its prefix. For example, if the string “bca” is in the dictionary, there will be nodes storing “bca”, “bc”, “b”, and white space “ ”. Nodes in the dictionary are blue, and those not in a dictionary are grey.

There are three types of arcs in the tree that indicated the relationship among the nodes: the child arc, the prefix arc, and the dictionary prefix arc.

The black arcs pointing from one node to its children is called a “child” arc, and the blue arc pointing from every node to its longest possible strict prefix in the graph is called “prefix” arc. The green arcs are “dictionary prefix” arc pointing from one node to the first blue node that can be reached by blue arcs.

When the algorithm reaches a node, it will output all the dictionary elements which ends at the current position in the input text by printing every node reached by following the dictionary prefix arcs, starting from that node, and continue until it reaches a node with no dictionary prefix arc linked to any other node.Particularly, if the node itself is a dictionary element, it will get printed.

3.2.2 Application and Limitation of Aho-corasick in Finding ORFs

In most cases, there will be a start codon at the beginning of ORF, and a stop codon at the end of ORF. Since Aho-Corasick can locate all the specified pattern simultaneously, intuitively, it is reasonable to find the ORFs by looking for all the fragments begin with start codon, and end with stop codon.

ORFM, a pipeline applied Aho-Corasick algorithm invented by *Ben J. Woodcroft, Joel A. Boyd, and Gene W. Tyson*.[6] ORFM is designed to deal with metagenomics data and search for ORFs at a high speed. According to their research, ORFM can find identical ORFs to similar tools like GetORF or Translate, but is 4 to 5 times faster.

However, the genome structure is much more complicated than we expected. On the one hand, a fragment with start codon at the beginning and stop codon at the end is not one-hundred percent ensured to be an ORF. On the other hand, not all the ORFs begin with general start codon. Even though ORFM can deal with huge amount of data and find all the fragments with specified patterns with a fast speed, not all the fragments it found are “true” ORFs.

To improve the precision of ORF annotated by pipeline, on the basis of ORFM, and another pipeline, ORF-finder[7]. For creating our pipeline, first, we combine ORFM and ORF-finder together, and adopted several limitations to produce a new ORF-predicted pipeline. The data preprocessing step of RiboCode is also embraced by this pipeline for data preprocessing.
Several limitations have also been added to this pipeline to ensure that this pipeline can find ORFs with comparatively high precision and sensitivity, while keep the advantage of fast speed. Here are the limitations added to the pipeline:

1. Instead of directly looking for ORFs in whole genome data, we used transcriptome prepared by the prepare_transcripts.py of RiboCode as the original input for this pipeline.
2. Choose the minimum length of ORFs as 96 nucleotides, since it’s the smallest length that allows all 6 reading frames to translate.
3. Add codon stringency in order to control the effects of fragments start from alternative start codons, or ORFs with no start codons. According to *ORF-finder manual*, it is recommended that the ORFs which start with an alternative codon must have the length of at least 49 codons, while the ORFs which start with no codons need to have the length of at least 100 codons.

3.3. Introduction to pipeline_part 2

The workflow of pipeline_part2 mainly consists of 2 parts: the data preprocessing step and the ORF-predicting step.

(a). Data preprocessing:

User can choose to start at the very beginning as noted in the first part of pipeline, or directly grab the data after finishing running STAR. Using prepare_transcripts.py in RiboCode to create the annotated transcripted data as the input of out pipeline.

(b). ORF-predicting:

After feeding the annotated transcripts fasta file to pipeline, the pipeline will automatically looking for ORF candidates. For the human genome data, it took around 10 minutes to finish the searching and mapping process.

All of the scripts mentioned above was integrated in a single python file named "pipeline_part2.py". This pipeline can be run using a single command line as:

```
python pipeline_part2.py
```

3.4. Results

In all, the revised ORF_finder, still based on Aho-Corasick algorithm but with the limitations mentioned before, showed good performance when applied to whole genome data. We tested the running time, ORFs-finding performance of this new pipeline, and finally evaluated the results with the predictions given by RiboCode.

3.4.1. Running Time Comparison

In this part, we compared the running time of ORFM, our pipeline and RiboCode when dealing with the same data, the human genome annotated transcripts produced by RiboCode prepare_transcripts.py. ORFM can finish the work in around 36 seconds, while our pipeline, the ORF-finder finished the task in 11 minutes. It took RiboCode around 30 minutes to finish. The time of data preprocessing is not concluded here.

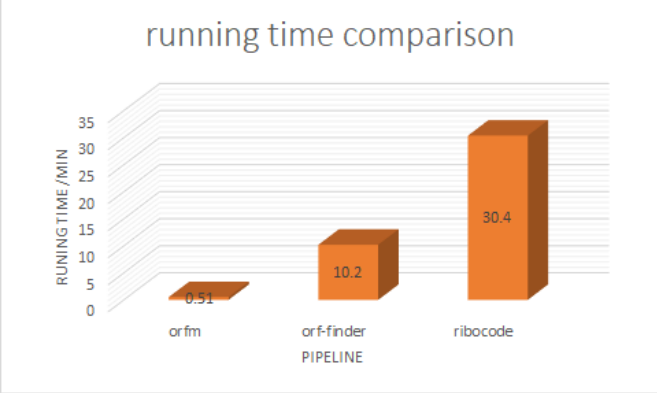


Figure 7. Running time comparison

3.4.2 Performance Comparison

Since the ORF-Finder doesn't find ORFs by probability distributions as RiboCode or other tools, such as RpBp, RiboORF or ORF-RATER, it is really hard to evaluate its performance.

From Xiao's paper, we can notice that when it comes to comparing the performance of different pipelines, instead of make comparison directly, the authors used simulated data and true data for every pipeline and make predictions, and acquire the statistical value for determine an ORF from every pipeline, compare the results predicted from true data and simulated data. For every pipeline, the performance is represented by comparing with itself on different data rather than compare with other pipeline.

Since there is no statistical value created by ORF-finder when finding ORFs, one way to evaluate the performance of ORF-finder is to compare the predictions it made with the predictions made by RiboCode when using the same data. However, since the output format of ORF-finder and RiboCode differ significantly from each other, comparing every nucleotide of all ORFs predicted by these two pipelines and expecting a perfect match is not very reasonable.

Nevertheless, we can still evaluate the performance of ORF-finder by comparing it to the results of RiboCode when the requirement is relaxed, such as we find the similarity between the results of ORF-finder and RiboCode rather than expecting a perfect match. We used a python package called difflib to calculate the similarity for every ORF pairs predicted respectively by ORF-Finder and RiboCode.

Difflib can make comparisons of two strings, and return a "ratio" parameter indicating the similarity between them. Generally speaking, the "ratio" indicates the numbers of operations, such as deletion, insertion, inversion and interchange, needed to make these two strings the same. The higher the ratio is between two strings, the similar these two strings are.

Compute the similarity of ORFs predicted by ORFM and ORF-finder with RiboCode predictions, choose ratio = 0.8 as the threshold value and assume the ORFs which has similarity higher than 0.8 are "good" predictions made by ORF-finders, since they are very similar to the predictions made by RiboCode and For name simplicity, we just call them the "true" ORFs.
We also evaluated the predictions made by ORFM in the same way as evaluating the performance of ORF-finder. As shown in figure 8, ORFM found 38769 ORF candidates in total, while only 9534 of its findings are "true" ORFs. Meanwhile, the ORF-finder found 16455 ORFs, with 9666 ORFs are regarded as "true" ORFs.

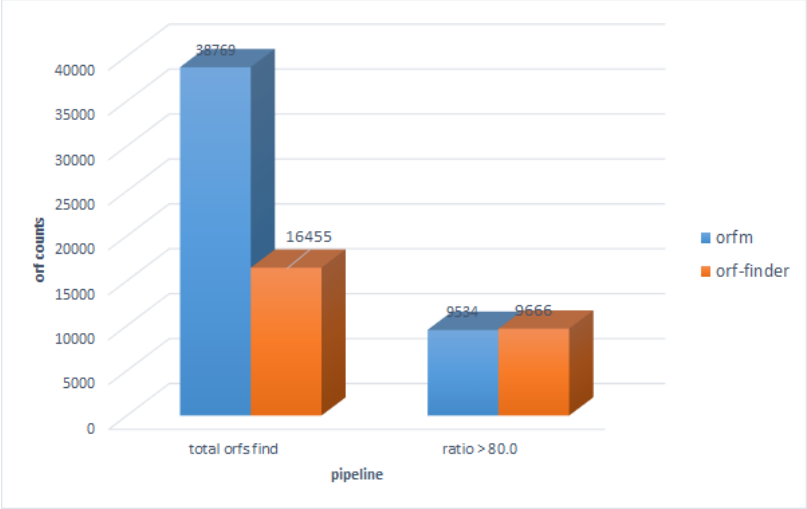


Figure 8. Candidate ORFs vs "True" ORFs

If we represent the data in figure 8 in another style, we can see the "true" ORFs only accounts for 24% of the ORFs it found. After we refine it by embracing several limitations, the percentage of "true" ORFs increased to 58%.

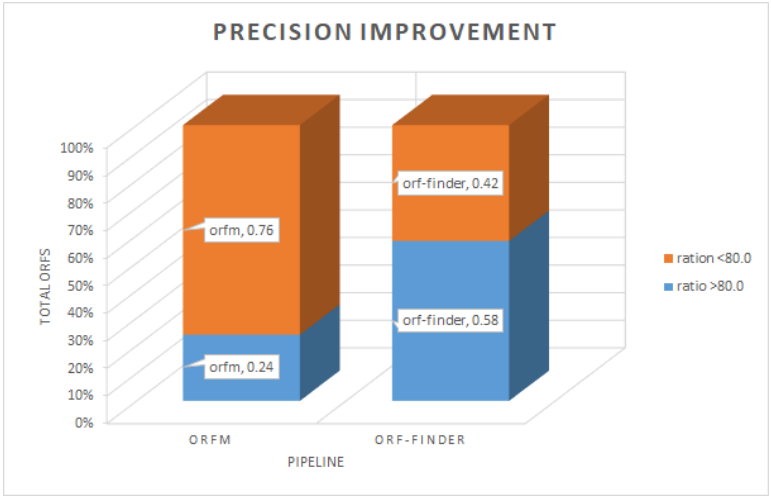


Figure 9. The composition of ORFs found by ORFM and ORF-finder

3.5 Analysis of ORF-finder

After combining those limitations to the ORF-finder, the predicting precision improved evidently, and it also sustains its advantage of finding ORFs candidates at a fast speed. As an string-searching algorithm looking for candidate ORFs, ORF-finder has higher efficiency than other ORF-finding tools depending on string-searching.

However, there are still potential drawbacks of ORF-finder. First, since ORF-finder only depends on the length limitation and the codon stringency, it might at first find much more ORFs than we expected, and then delete true ORFs when manage to correct this inaccuracy.

Second, since ORF-finder doesn't use any data which can generate useful probability distribution to make predictions like ribosome profiling data, it lacks a standard to evaluate the precision of ORF-finder. The similarity calculated by comparing pairs of ORF fragments can only partially reflect the improvement in making predictions, but can not be regarded as the proof of the precision of ORF-finder.

4. Discussion

4.1 Comparison with Group A

4.1.1 Similarities:

Group A also use RiboCode as base-software.

4.1.2 Difference:

Group A uses tailored experimental reference transcript file, rather than files from public database. This change is helpful for overcoming the less specificity of annotation samples under different conditions. Also, they used incorporated additional tools for RNA-seq alignment and transcript assembly, the HiSAT2 and StringTie.

The pipeline results are not included in their manual folder, so detailed comparison can not be made.

4.2 Comparison with Group B

Their method is supervised learning, while our uses de-novo approach.

Advantage: their method can find short ORFs, but ours can't.

Disadvantage: their method needs pre-annotated training set which may contain bias.

4.3 Comparison with Group D

4.3.1 Similarities:

Based on Group D's manual description, their pipeline is built on top of RiboCode, which is similar to our implementation.

4.3.2 Difference:

Instead of implement the entire RiboCode, Group D are focusing on improving the quality of one input file, the GTF input file, using Augustus, in hope to getting better result from RiboCode. Moreover, our validation method is very different. We use Blast to validate the significance of ORFs. They use RiboCode to validation the result from Augustus.

Group D has also ran their pipeline with mouse data, and their RiboCode result is very similar with ours. This imply that using Augustus's pre-trained hidden markov model to annotate genomic sequence may not improve RiboCode's result significantly on given mouse data. More experiments are required to testing the utility of Augustus.

4.4 Comparison between the RiboCode result of human and mouse

4.4.1 Similarity: Majority of ORFs found are annotated.

4.4.2 Difference: There are more novel ORFs in human genome than mouse genome. Also, the total number of ORFs found in human is larger than that of mouse. This also makes sense because human genome is more complex than mouse's.

4.5 Advantage and disadvantage of RiboCode

4.5.1 Advantage: efficient, unbiased, user-friendly, require little computation resource, can identify overlapping ORFs.

4.5.2 Disadvantage: no validation

The annotation result produced by RiboCode, as our base algorithm, heavily relies on ribosome profiling data and the 3-nt periodicity in it, while other informations implied by ribosome profiling data might be underrated or neglected. Moreover, when using ribosome profiling data, though the RiboCode package accounts for overlapping open reading frames, it might still discard information that is found only in polysome profiles.

4.6 Areas of Improvement

As stated in the description of our pipeline, the original goal of our pipeline is to incorporate the results from various ORF prediction approaches. Traditional methods for predicting ORFs includes De novo approach and Homology-based approach. RiboCode heavily depends on the ribosomal profiling data, which is a new approach for predicting ORFs. If time allows, we would like to include some other popular softwares. For example, commonly used softwares for De novo annotation includes Augustus, Genscan, and Glimmer; whereas available software for Homology-based annotation includes Genewise. We can further cross-reference the results and obtain the overlapping regions.

The second part of our pipeline has implemented a variant of traditional De novo approach. The next step would be comparing the final result from both parts and report all ORFs found with an appropriate confidence score. The confidence score can be defined based on the weighted result from each part, i.e., ORF has high confidence score if reported from both approaches.

Although we integrate BLAST into our pipeline, the running time for BLAST is very long. We can further improve this by finding some software that has higher efficiency.

References

1. Xiao, et.al, De novo annotation and characterization of the translatome with ribosome profiling data, February 28, 2018, Nucleic Acid Research
2. Wikipedia, https://en.wikipedia.org/wiki/De_novo_transcriptome_assembly.
3. Bowtie Manual: http://bowtie-bio.sourceforge.net/manual.shtml#the-bowtie-build-indexer
4. https://www.illumina.com/techniques/sequencing/rna-sequencing/ribosome-profiling.html
5. https://en.wikipedia.org/wiki/Gene_transfer_format
6. Ben J. Woodcroft, Joel A. Boyd, Gene W. Tyson; OrfM: a fast open reading frame predictor for metagenomic data, *Bioinformatics*, Volume 32, Issue 17, 1 September 2016, Pages 2702–2703
7. ORF-Finder: https://github.com/zkstewart/orf-finder-py