Final Practical Report
Group number: 2
Group members: Maximilian Senftleben, Zhong Hao Daryl Boey

Summary: In this final assignment, 3 original scripts (and many more) were written to support the aim of locating genes and reconstructing phylogenic trees from 5 unknown genomes that were assigned to the group. The located ORFs were counted, as well as compared with reference genomes from UniProt, to test the performance of the ORF finder. The genomes were also analyzed for various frequencies, and these frequencies used to compute evolutionary distances, which were then used to reconstruct phylogenic trees.

1. Summary of your five genomes incl families, species, chromosomes, genome length.

File name	Family	Species	Chromosomes	Genome length	Organism type
16.fa.txt	Planctomycetaceae	*R. baltica* SH1	?	7149689	Prokaryotic
20.fa.txt	Thermotogaceae	*T. maritima* strain Tma100	Complete Genome	1869610	Prokaryotic
29.fa.txt	Saccharomycetaceae	*S.cerevisiae* S288C	IX	439888	Eukaryotic
44.fa.txt	Leuconostocaceae	*L. gelidum* JB7	Complete Genome	1893500	Prokaryotic
47.fa.txt	Neisseriaceae	*N. meningitidis* alpha710	Complete Genome	2242948	Prokaryotic

Table 1. Summary of genomes.

2. The results of applying your three Python scripts to your five genomes:
   a. The GC, nucleotide and dinucleotide frequencies for the five genomes & amino acid and diamino acid frequencies for your predicted proteins (output of your ORF finder) using your statistics tool.

The python script stat_tool_1.py was used to evaluate both GC, nucleotide, and dinucleotide frequencies for the 5 genomes.

genome: 16.fa.txt	genome: 20.fa.txt	genome: 29.fa.txt	genome: 44.fa.txt	genome: 47.fa.txt
GC-content: 0.5548	GC-content: 0.4625	GC-content: 0.3890	GC-content: 0.3668	GC-content: 0.5169
#A = 1589939/7149689	#A = 504109/1869610	#A = 134339/439888	#A = 600998/1893500	#A = 540971/2242948
#C = 1981614/7149689	#C = 426011/1869610	#C = 85465/439888	#C = 346142/1893500	#C = 577262/2242948
#T = 1592923/7149689	#T = 500833/1869610	#T = 134423/439888	#T = 598020/1893500	#T = 542592/2242948
#G = 1985204/7149689	#G = 438657/1869610	#G = 85661/439888	#G = 348339/1893500	#G = 582122/2242948
#N = 9/7149689	#N = 0/1869610	#N = 0/439888	#N = 0/1893500	#N = 0/2242948
#AA = 359096/7149689	#AA = 119138/1869610	#AA = 34423/439888	#AA = 159766/1893500	#AA = 136187/2242948
#AC = 421691/7149689	#AC = 100392/1869610	#AC = 23399/439888	#AC = 105514/1893500	#AC = 116880/2242948
#AT = 387766/7149689	#AT = 112580/1869610	#AT = 38074/439888	#AT = 190601/1893500	#AT = 136676/2242948
#AG = 329767/7149689	#AG = 129961/1869610	#AG = 26070/439888	#AG = 88325/1893500	#AG = 97600/2242948
#CA = 489129/7149689	#CA = 110613/1869610	#CA = 28585/439888	#CA = 138243/1893500	#CA = 140876/2242948
#CC = 388521/7149689	#CC = 80193/1869610	#CC = 14956/439888	#CC = 54062/1893500	#CC = 121349/2242948
#CT = 330792/7149689	#CT = 127257/1869610	#CT = 25992/439888	#CT = 87166/1893500	#CT = 96573/2242948
#CG = 704454/7149689	#CG = 92039/1869610	#CG = 13236/439888	#CG = 58970/1893500	#CG = 195774/2242948
#TA = 105255/7149689	#TA = 67219/1869610	#TA = 31581/439888	#TA = 149211/1893500	#TA = 83819/2242948

#TC = 543628/7149689	#TC = 160752/1869610	#TC = 27250/439888	#TC = 94947/1893500	#TC = 125051/2242948
#TT = 359547/7149689	#TT = 117679/1869610	#TT = 34410/439888	#TT = 159111/1893500	#TT = 136887/2242948
#TG = 493028/7149689	#TG = 113839/1869610	#TG = 28797/439888	#TG = 138835/1893500	#TG = 143111/2242948
#GA = 544841/7149689	#GA = 165101/1869610	#GA = 27377/439888	#GA = 96986/1893500	#GA = 126462/2242948
#GC = 559056/7149689	#GC = 68766/1869610	#GC = 17163/439888	#GC = 83918/1893500	#GC = 191292/2242948
#GT = 423352/7149689	#GT = 101972/1869610	#GT = 23563/439888	#GT = 105227/1893500	#GT = 118731/2242948
#GG = 389509/7149689	#GG = 85445/1869610	#GG = 14961/439888	#GG = 54283/1893500	#GG = 122793/2242948

Table 2. Nucleotide content of genomes.

Proteome: 16.fa.txt.pfa Amino acid frequency:	Proteome: 22.fa.txt.pfa Amino acid frequency:	Proteome: 29.fa.txt.pfa Amino acid frequency:	Proteome: 44.fa.txt.pfa Amino acid frequency:	Proteome: 47.fa.txt.pfa Amino acid frequency:
#A = 201732/2136223	#A = 34721/591793	#A = 6050/107024	#A = 43737/543724	#A = 63433/629939
#C = 23629/2136223	#C = 4060/591793	#C = 1372/107024	#C = 1137/543724	#C = 6710/629939
#D = 137859/2136223	#D = 29409/591793	#D = 6057/107024	#D = 30949/543724	#D = 32866/629939
#E = 133128/2136223	#E = 52921/591793	#E = 7014/107024	#E = 28192/543724	#E = 38300/629939
#F = 77611/2136223	#F = 30824/591793	#F = 4895/107024	#F = 24271/543724	#F = 26326/629939
#G = 162099/2136223	#G = 40986/591793	#G = 5233/107024	#G = 35566/543724	#G = 48429/629939
#H = 47555/2136223	#H = 9341/591793	#H = 2149/107024	#H = 11531/543724	#H = 14054/629939
#I = 107341/2136223	#I = 42568/591793	#I = 6782/107024	#I = 43306/543724	#I = 36567/629939
#K = 73760/2136223	#K = 44977/591793	#K = 7751/107024	#K = 32761/543724	#K = 35954/629939
#L = 201366/2136223	#L = 59554/591793	#L = 9990/107024	#L = 53355/543724	#L = 61782/629939
#M = 49888/2136223	#M = 13521/591793	#M = 2171/107024	#M = 14905/543724	#M = 15093/629939
#N = 72372/2136223	#N = 21384/591793	#N = 6564/107024	#N = 28408/543724	#N = 25907/629939
#P = 111032/2136223	#P = 23554/591793	#P = 4721/107024	#P = 18236/543724	#P = 26906/629939
#Q = 86631/2136223	#Q = 11932/591793	#Q = 4427/107024	#Q = 25038/543724	#Q = 25467/629939
#R = 140785/2136223	#R = 32590/591793	#R = 4670/107024	#R = 20988/543724	#R = 35348/629939
#S = 151835/2136223	#S = 33353/591793	#S = 9846/107024	#S = 33163/543724	#S = 35174/629939
#T = 124754/2136223	#T = 26779/591793	#T = 6705/107024	#T = 34858/543724	#T = 33063/629939
#V = 153409/2136223	#V = 51533/591793	#V = 5992/107024	#V = 39122/543724	#V = 42120/629939
#W = 32759/2136223	#W = 6553/591793	#W = 1099/107024	#W = 5733/543724	#W = 7469/629939
#Y = 46674/2136223	#Y = 21233/591793	#Y = 3536/107024	#Y = 18468/543724	#Y = 18971/629939

Table 3. Amino acid content of genomes.

Diamino acid frequencies included as attached files, diamino_out_*.txt.

i. Present the formulas you used for the GC content and frequency calculation.

The formula used for GC content was (seq.count("C") + seq.count("G"))/len(seq). The formula used for frequency calculation was seq.count(nucl)/len(seq), with nucl being an iterative element for a defined set of nucleotides/ dinucleotides. The algorithm for combination counts such as the dinucleotide frequency is however slightly different, owing to the fact that for a sequence of 10 characters there are 9 possible combinations. After a quick count we found that N (undefined nucleotides) were only found in 16.fa.txt in few numbers (0.000126%), and as such did not account for it in frequency calculations.

$$Frequency = \frac{count(nucleotide\ or\ aminoacid)}{len(total)}$$

$$Frequency = \frac{count(dinucl.\ or\ diaminoa.)}{len(total) - 1}$$

ii. If applicable use charts for the results you obtained.

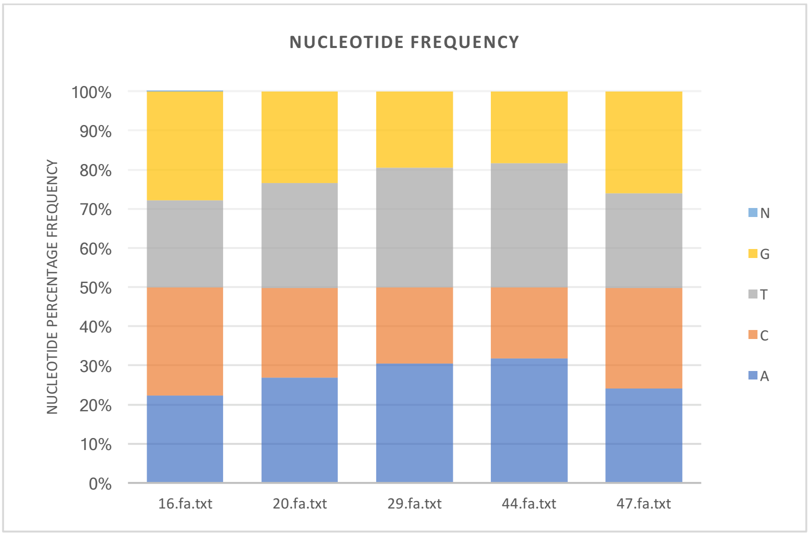

Fig. 1. Nucleotide frequencies of genomes.

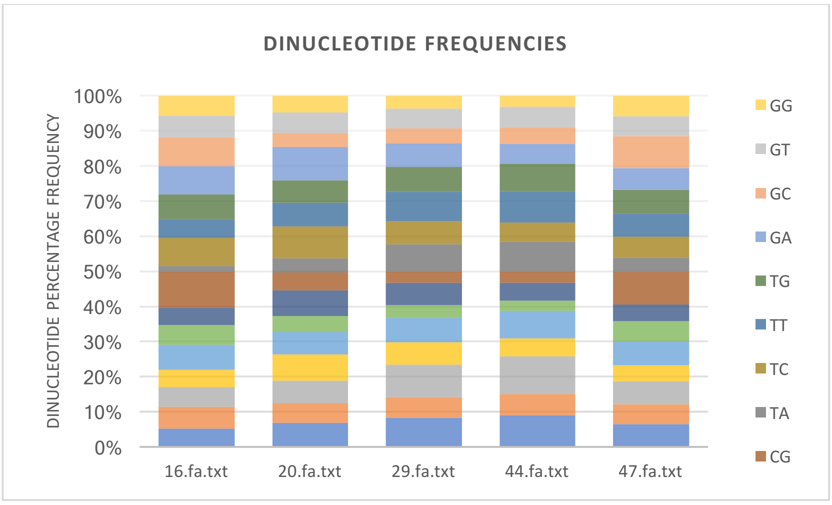

Fig. 2. Dinucleotide frequencies of genomes.

b. The predicted ORFs in your five genomes by your ORF predictor (include some examples and some statistics, e.g.
i. Which basic assumption on ORFs did you apply?

The script orf_finder_2.py was written based on the conditions stated below, with an example output file called out_orf16.fasta.

The basic assumption of an Open Reading Frame (ORF), is that it is a stretch of DNA with a start codon on the 5' end, and a stop codon on the 3' end[i], which has the possibility to encode a protein coding sequence[ii]. Therefore, our ORF finder searches for sequences flanked by a start codon and stop codon, across all 6 reading frames; 3 per strand, forward and reverse. We also assumed that prokaryotic ORFs would be in general longer than 300bp[iii], while eukaryotic ORFs are more complex due to splicing and therefore assigned a length between 100-1500bp.

The initial gene turnout was extremely high, as such an additional condition was included in the ORF finder script to remove overlapping genes; ie., a shorter ORF nested within a longer. This resulted in a more realistic number of genes, except for the eukaryotic genome. We attribute the inaccuracy of the method to the lack of accounting for splicing and introns within the eukaryotic genome.

The code was written with several nested for-loops. First, the script is verified to be accounting for three different reading frames. Each reading frame was scanned for an ATG and then, starting from there and ending at a stop-sequence, with the found ORF added to a list. By implementing a variable (t2), which is set to the iterating variable after appending an ORF, the overlaps can be removed. The script will always take the longer sequence, e.g. the first ATG to the next stop-sequence. The same was done for the reverse strand, where the sequence was first translated into its complement and then mirrored. Then the same script was used as above, as it reads in the sequence in 5′→3′.

ii. How many ORFs per genome did you predict?

File	Gene count from reference	Gene count from ORF finder
16.fa.txt	7406	11,420
20.fa.txt	5806	2,725
29.fa.txt	232	1,573
44.fa.txt	1967	1,739
47.fa.txt	2104	3,847

Table 4. ORFs counted using ORF finder compared against reference (UniProt).

iii. Assess the accuracy of your ORF predictor by comparing its predictions to a reference. Specify the used reference, accuracy measure, criteria for defining true positives, false positives, false negatives, etc..

For accessing the performance, the F1-score was used. As a reference, the proteomes of the organisms were downloaded from Uniprot. For the query sequence, the ORFs of the organisms were translated with Biopython into amino acid sequences. Then, the true positives, false positives and the false negatives were calculated by comparing the protein-coding genes of the reference and query proteome. The comparisons were compiled into the following statistics.

Reviewing the F1-scores for the various genomes, 20.fa.txt and 44.fa.txt have the healthiest scores of 0.43 and 0.72 respectively, while the other 3 genomes performed relatively poorly. The worst offending genome was 16.fa.txt, with an extremely low score of 0.042, which we attribute to an issue with the original genome, which had an enormous size of over 7 * 10^6 base pairs. BLASTing this genome showed multiple perfect hits for various segments of the genome, but none were complete. As such we were unable to obtain a suitable Uniprot reference file for the genome, resulting in the low F1-score.

		Predicted Class		**F1-score**	0.0418
	16.fa.txt	P	N	**Len Pred**	11420
Actual Class	P	391	6880	**Len Ref**	7271
	N	11029	NIL		

		Predicted Class		**F1-score**	0.4274
	20.fa.txt	P	N	**Len Pred**	2725
Actual Class	P	978	874	**Len Ref**	1852
	N	1747	NIL		

		Predicted Class		**F1-score**	0.2519
	29.fa.txt	P	N	**Len Pred**	730
Actual Class	P	119	96	**Len Ref**	215
	N	611	NIL		

		Predicted Class		**F1-score**	0.7202
	44.fa.txt	P	N	**Len Pred**	1739
Actual Class	P	1273	523	**Len Ref**	1795
	N	466	NIL		

		Predicted Class		F1-score	0.0899
	47.fa.txt	P	N	Len Pred	3847
Actual Class	P	263	1742	Len Ref	2001
	N	3584	NIL		

Tables 5 – 9. Analysis scores and F1 scores for metric comparisons.

iv. Which additional improvements would be possible to increase the accuracy of your ORF predictor?

The first improvement that comes to mind will involve improving the specificity of the ORF predictor, by utilising more information available in the literature involving transcription and translation. This could include requiring that ORFs have upstream promoter sequences like the Pribnow box (TATAAT)[iv], or Shine-Delgarno sequence (AGGAGG)[v]. We briefly tried implementing these specifications, which resulted in a drastic reduction in the number of ORFs found, much lower than the reference. The suspected cause is the script being overly specific for the upstream positions where the promoter sequences ought to be located. Given more time a script can be written to compensate for a larger range of such upstream positions.

Another commonly used method to increase the accuracy of the finder is the Hidden Markov Model (HMM) architecture, found in the hugely popular GENSCAN[vi], which overcomes the complexity of splicing in eukaryotic genomes. An additional way to test and benchmark the ORFfinder would be to include metrics using from GLIMMER as a comparison in addition to the UniProt databases.

c. The distance matrix for your genomes computed using your third script with the output of your first script.

   i. Which distance method did you use and why?

      The distances matrices were derived using the script dist_matr_3.py, producing *.grimm files as output.

      A variety of distance based methods were applied, the first was based on the pure distance between the various GC percentage values of the gene sets, the second based on distances between nucleotide frequencies, and the third based between dinucleotide frequencies. We sought to include a variety of distance calculation methods to compare the impact it may have on resulting trees.

   ii. Construct and present a phylogenetic tree based on the calculated distance matrix. Which tree building method did you use and why?

The trees were constructed using Belvu, using the Scoredist distance correction method, with Neighbour Joining (NJ) used for tree construction, producing an unrooted tree. < belvu –T R *inputfile.grimm*>

The alternative method to using NJ would be Unweighted Pair Group Method with Arithmetic Mean (UPGMA), but this method assumes a constant evolutionary rate, and is not suitable for our purposes as we have not verified if all 5 organisms share this similar rate of evolution.

iii. Compare shortly this tree with the trees you created during the various practicals.

We include a tree produced using the distance calculated with dinucleotide frequencies, normalized by multiplying all values by 1000, to adjust for the initial extremely miniscule values. Other attempts at normalizing the distances include using preprocessing modules from scikitlearn, such as normalize (both l1 and l2) resulted in inconsistent changes of values in the array that should have been identical, such as 16.fa.txt vs 20.fa.txt, and 20.fa.txt vs 16.fa.txt.

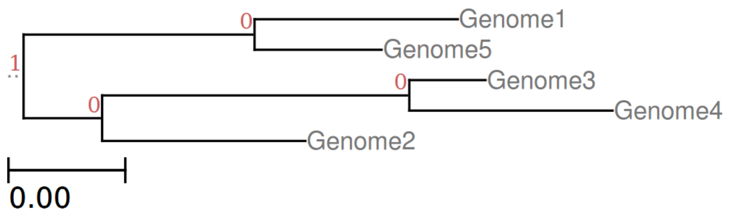

Fig. 3. Phylogenic tree produced using dinucleotide frequencies.

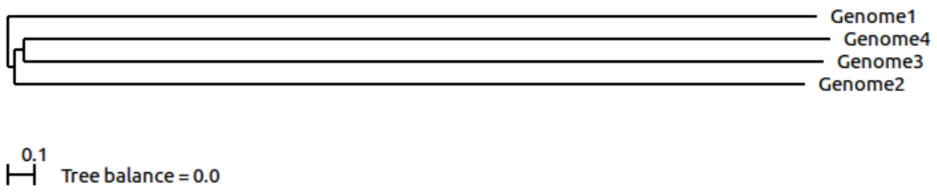

Fig. 5. Phylogenic tree produced using GRIMM-inferred distances.

Compared to the tree produced in Practical 5 with GRIMM, the basic branching structure is almost identical, except for the addition of genome 5, as we've included the eukaryotic genome (29.fa.txt) in this phylogenic reconstruction. The gene lengths also vary between the 2 trees, due to the differences in distance calculation, such as GRIMM vs dinucleotide frequency derived distances.

References

[i] Crick, F. (1968). The origin of the genetic code. Journal Of Molecular Biology 38, 367-379.

[ii] Zvelebil, M., and Baum, J. (2008). Understanding bioinformatics (New York: Garland Science).

[iii] Burge, C. B. and Karlin, S. (1998) Finding the genes in genomic DNA. *Curr. Opin. Struct. Biol.* **8,** 346-354.

[iv] Pribnow, D. (1975). Nucleotide sequence of an RNA polymerase binding site at an early T7 promoter. Proceedings Of The National Academy Of Sciences *72*, 784-788.

[v] Malys, N. Mol Biol Rep (2012) 39: 33. https://doi.org/10.1007/s11033-011-0707-4.

[vi] Burge, C. and Karlin, S. (1997) Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.* **268,** 78-94.