Department of Electronic and Telecommunications Engineering
University of Moratuwa

BM4322 - Genomic Signal Processing

Assignment

Rajapaksha N.N. - 210504L

1 Introduction

Promoter detection is a fundamental task in understanding gene regulation and transcription initiation mechanisms in prokaryotes. In bacteria, RNA polymerase relies on the σ^2 subunit to recognize specific promoter sequences that mark transcription start sites. According to Liu et al. (2011), the σ^2 subunit exhibits a strong affinity toward promoter motifs of the form `WAWWWT`, where `W` represents either adenine (A) or thymine (T). These motifs are typically located approximately ten bases upstream of the transcription start site (TSS).

In this assignment, the goal was to computationally detect these promoter motifs in a given bacterial genome using the principles of genomic signal processing. The exercise involved three major stages:

1. Extracting upstream regions of predicted genes and constructing a Position Probability Matrix (PPM) from manually identified promoter-like sequences.
2. Performing a statistical alignment across remaining upstream regions to predict promoter presence.
3. Validating the promoter detection model across genomes assigned to other students.

The methodology employed in this analysis combines biological domain knowledge with statistical modeling and computational tools such as Biopython and NumPy. By constructing the PPM and computing consensus-based alignment scores, the analysis quantitatively measures promoter similarity and frequency within the genome. The workflow followed in this study closely aligns with the procedure described in the assignment brief, ensuring both biological interpretability and computational rigor.

2 Promoter Region Selection & PPM Construction

2.1 Data Acquisition and Preprocessing

In the first step of the analysis, genomic sequence (FASTA format) and the genome annotation (GFF format) for the Genomes were downloaded using NCBI CLI. A custom class, `GenomeManager` (see Appendix 5), was defined to encapsulate these tasks: reading the annotation file, indexing genes, retrieving sequence features (chromosomes, strands, coordinates), and extracting the upstream regions for each gene.

```
ref_sequence = ref_genome.read_fna(sequence_itself=True)
gff_database = ref_genome.load_gff()
```

2.2 Filtering and Selecting genes

Then I loaded the gene sequences using the annotation in the `.gff` file for the given genome via the custom class `GenomeManager`. From this, I randomly selected 1,100 genes. Out of the 1,100 genes, 100 genes were selected to construct the PPM. For each of these genes, the region from 15 to 5 bases upstream of the gene start site was extracted using Biopython. On the positive strand, bases were directly extracted, while on the negative strand the reverse-complement of the corresponding downstream region was considered. If the region did not contain at least six consecutive W's (where W = A or T), it was rejected. From each accepted region, the six

bases most likely representing the promoter core were extracted and used to build the Position Probability Matrix (PPM).

```
N = 1100
regions = ref_genome.extract_promoter_regions(find_method="random", n=N)
all_idx = set(range(len(regions)))
ppm_idx = random.sample(list(all_idx), 100)
regions_ppm = [regions[i] for i in ppm_idx]
```

2.3 Constructing the PositionProbability Matrix (PPM)

From the upstream regions of the selected 100 genes, promoter sequences of the form `WAWWWT` were identified and extracted to construct the motif model.

```
promoters = []
for region in regions_ppm:
    promoter = find_promoter(region)
    if promoter:
        promoters.append(promoter)
```

A Position Probability Matrix (PPM) was then built using Biopython's `motifs` module, with heuristic pseudocounts added for under-represented bases (C and G) to avoid zero probabilities. After computing the PPM, the consensus sequence and its associated consensus-score were calculated to serve as the reference motif for subsequent alignment against genomic upstream regions.

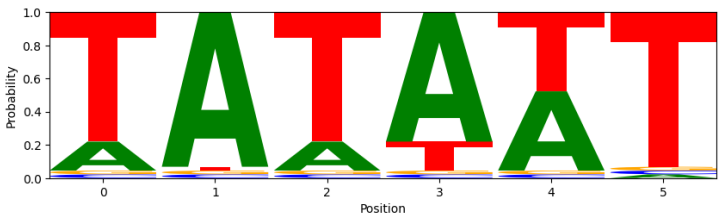
```
m = motifs.create(promoters, alphabet="ACGT")
ppm = m.counts.normalize(pseudocounts=0.15)
```

Here is the constructed PPM and the consensus.

```
Position Probability Matrix:
        0     1     2     3     4     5
A:   0.17  0.93  0.17  0.78  0.48  0.02
C:   0.02  0.02  0.02  0.02  0.02  0.02
G:   0.02  0.02  0.02  0.02  0.02  0.02
T:   0.78  0.02  0.78  0.17  0.48  0.93

Consensus promoter: TATAAT
```

(a) PPM and Consensus



(b) Sequence Logo for Consensus Promoter Motif

Figure 1: Comparison of motif-related visualisations.

The sequence logo summarises the nucleotide preferences across the identified 6-base promoter motif: adenine (A) and thymine (T) dominate most positions, forming the characteristic A/T-rich pattern of bacterial promoters. The conserved prominence of A and T highlights their critical role in promoter recognition and transcription initiation. The total height of each letter stack corresponds to the information content (in bits) at that position, reflecting the degree of conservation: taller stacks mark positions of high conservation and thus greater importance for motif recognition, while shorter stacks indicate greater variability and reduced significance. Within each stack, the height of a given letter is proportional to the frequency of that nucleotide (A, T, C or G) at that position, thereby encoding both enrichment and variability in a single visual.

3 Statistical Alignment & Promoter Prediction Results

From the upstream regions (-15 to -5 bases) of the selected 1 100 genes, alignment scores were calculated for each of the remaining 1 000 genes using the derived PPM. Statistical alignment was applied according to the following criterion in the given code snippet. The heuristic threshold was determined empirically by inspecting the histogram of align scores.

```
consensus_promoter = str(m.consensus)
consensus_score = alignment_score(ppm, consensus_promoter)
threshold_score = 1.8

regions_1000 = regions_non_ppm

ref_promoter_hits = []
ref_alignment_scores = []

for region in regions_1000:
    score = alignment_score(ppm, region)
    ref_alignment_scores.append(score)
    if score - consensus_score >= -threshold_score:
        ref_promoter_hits.append(region)
```

For the genome I was given under Question 2, namely `GCA_900636475.1`, a total of 29 promoters were detected out of the 1 000 upstream sequences analysed.

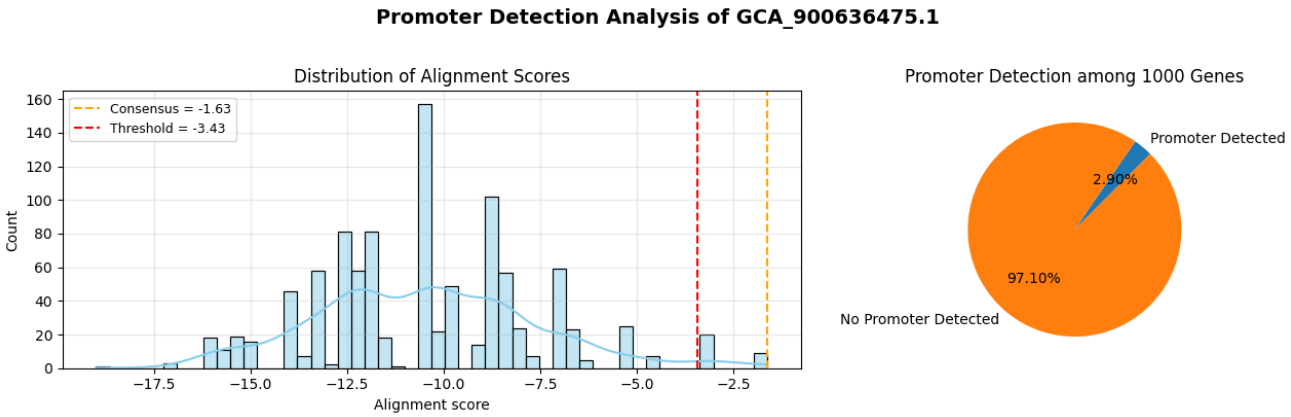
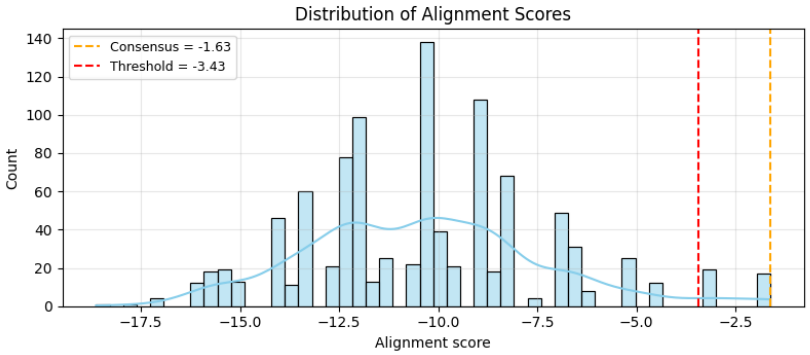


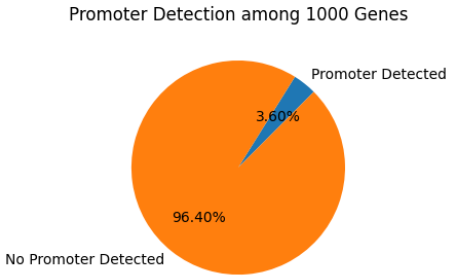
Figure 2: Alignment Scores and Promoters Detected from $GCA_900636475.1$

4 Statistcal Alignment for other 5 genomes

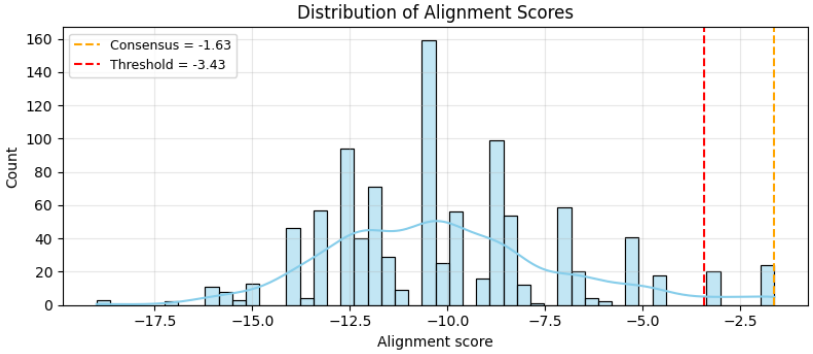
Using the PPM obtained from the first genome, statistical alignment was carried out on the 1,000 randomly selected upstream regions from each of the five additional genomes. The following present the results obtained from this cross-genome validation.

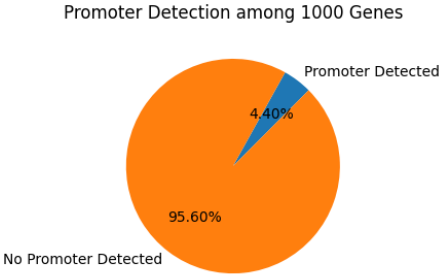
Promoter Detection Analysis of GCA_001457635.1



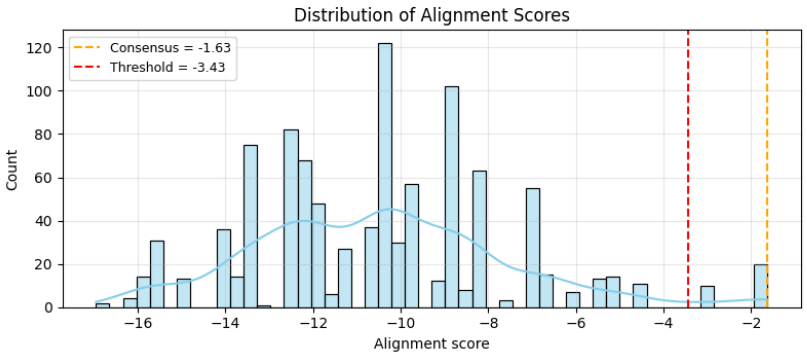


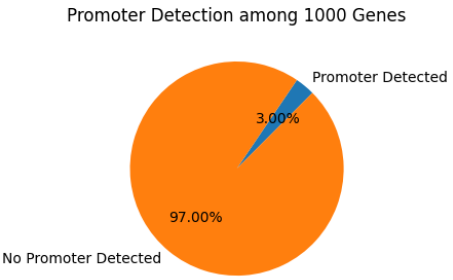
Promoter Detection Analysis of GCA_019048645.1

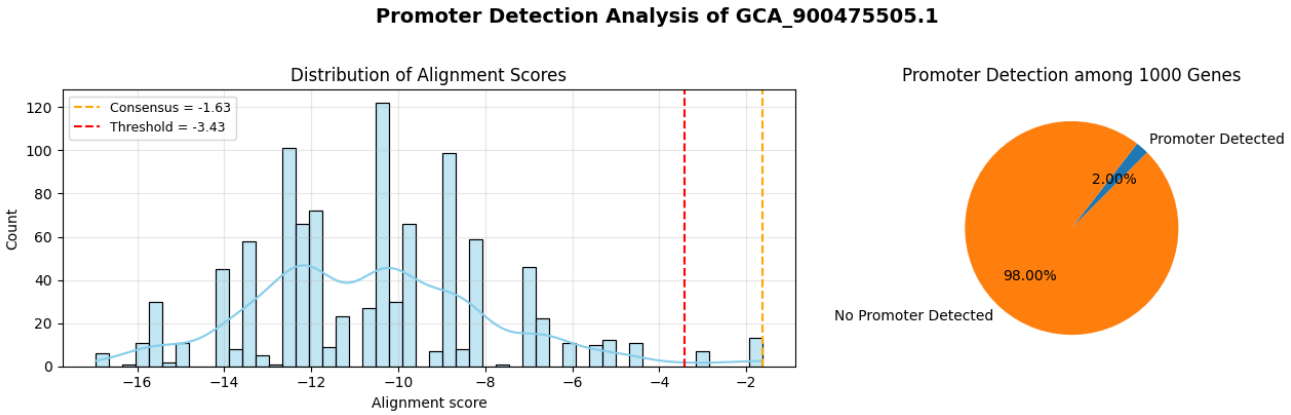




Promoter Detection Analysis of GCA_900637025.1







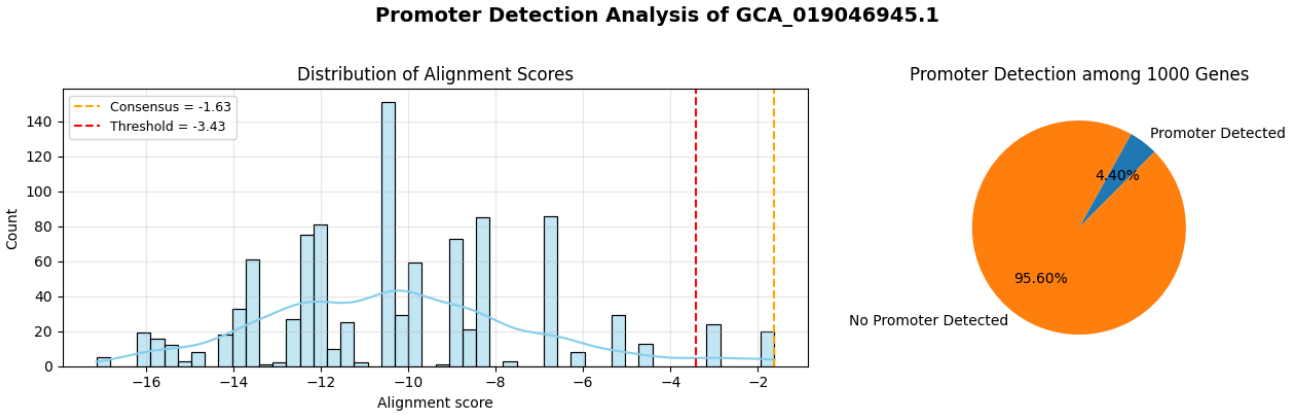


Table 1: Summary of promoter detection results across other 5 genomes.

Student	Genome	Promoters Detected
210079K	GCA_001457635.1	36
210179R	GCA_019048645.1	44
210657G	GCA_900637025.1	30
210707L	GCA_900475505.1	20
210732H	GCA_019046945.1	44

5 Conclusion

In this study, I developed a streamlined pipeline for detecting σ^2-subunit-dependent promoters in bacterial genomes. I started by using the `GenomeManager` class to extract upstream regions of annotated genes, then selected 100 high-confidence promoter candidates from 1,100 genes and constructed a Position Probability Matrix (PPM) reflective of the core motif. Next, I applied this PPM in sliding-window statistical alignment to approximately 1,000 upstream sequences within the same genome, successfully identifying putative promoter regions. Finally, I extended the approach to five additional genomes, confirming that the motif model was transferable beyond the initial dataset. The identified motif showed the expected A/T-rich signature and demonstrates the practical value of combining domain knowledge with probabilistic modelling

for promoter prediction. Although the current framework remains relatively simple and lacks experimental validation, it offers an interpretable and reproducible starting point for broader bacterial promoter screening and sets the stage for more advanced integration of complex regulatory features and transcriptomic data.

Appendix

A. GenomeManager Class Implementation

```python
class GenomeManager:
    def __init__(self, base_path: str, genome_id: str):
        """
        base_path: root directory containing all genome subfolders (e.g. "data/ncbi_dataset/data")
        genome_id: name of the genome folder (e.g. "GCA_900636475.1")
        """
        self.genome_dir = os.path.join(base_path, genome_id)
        self.fna_path = None
        self.gtf_path = None
        self.gbff_path = None
        self.gff_path = None
        self._find_files()

    def _find_files(self):
        """Scan the genome directory and assign file paths for .fna, .gtf, .gbff, .gff"""
        for fname in os.listdir(self.genome_dir):
            lower = fname.lower()
            full = os.path.join(self.genome_dir, fname)
            if lower.endswith(".fna") or lower.endswith(".fa") or lower.endswith(".fasta"):
                self.fna_path = full
            elif lower.endswith(".gtf"):
                self.gtf_path = full
            elif lower.endswith(".gbff") or lower.endswith(".gbk") or lower.endswith(".gbff"):
                self.gbff_path = full
            elif lower.endswith(".gff") or lower.endswith(".gff3"):
                self.gff_path = full

    def summary(self):
        """Print what files are detected in this genome folder."""
        print(f"Genome directory: {self.genome_dir}")
        print(f"  FASTA (.fna): {self.fna_path}")
        print(f"  GTF (.gtf):   {self.gtf_path}")
        print(f"  GBFF (.gbff): {self.gbff_path}")
        print(f"  GFF (.gff/.gff3): {self.gff_path}")

    def read_fna(self, sequence_itself: bool = True):
        """Read the FASTA (.fna) file, return list of SeqRecord objects."""
        if not self.fna_path:
            raise FileNotFoundError("No .fna file found in genome directory.")
        genome = list(SeqIO.parse(self.fna_path, "fasta"))

        if not len(genome) == 1:
            raise ValueError(f"Expected 1 sequence in .fna file, found {len(genome)}")

        seq = genome[0]
```

```
46
47          if sequence_itself: return seq.seq
48          else: return seq
49
50      def load_gff_from_gffutils(self):
51          """Load GFF file using gffutils."""
52          if not self.gff_path:
53              raise FileNotFoundError("No .gff file found in genome directory.")
54          # Implement GFF parsing logic here
55
56          gff_db = gffutils.create.create_db(data=self.gff_path, dbfn=':memory:', force=True)
57          return gff_db
58
59      def load_gff(self):
60          """Load GFF file using gffutils."""
61          if not self.gff_path:
62              raise FileNotFoundError("No .gff file found in genome directory.")
63          # Implement GFF parsing logic here
64
65          # Use read_csv with comment="#" so lines starting with \#" are ignored
66          gff_db = pd.read_csv(self.gff_path, sep="\t", comment="#", header=None,
67              names=["seqid", "source", "feature", "start", "end", "score", "strand", "phase",
                ↪  "attributes"],
68              dtype={
69                  "seqid": str,
70                  "source": str,
71                  "feature": str,
72                  "start": int,
73                  "end": int,
74                  "score": str,    # may be \." or numeric
75                  "strand": str,
76                  "phase": str,
77                  "attributes": str
78              },
79              # In case there are \bad lines" (e.g. wrong number of columns), skip them
80              on_bad_lines="skip"
81          )
82
83          return gff_db
84
85      def extract_promoter_regions(self, find_method: str = "random", n: int = 100,
        ↪  upstream_start: int = 15, upstream_end: int = 5):
86          """Extract promoter regions from fasta file based on GFF annotations."""
87
88          if find_method not in ["random", "first"]:
89              raise ValueError(f"Unknown find_method: {find_method}")
90
91          gff_db = self.load_gff()
92          genome_seq = self.read_fna(sequence_itself=True)
93
94          # Filter for gene features
95          genes = gff_db[gff_db['feature'] == 'gene']
96          if genes.empty:
97              raise ValueError("No gene features found in GFF file.")
98          elif len(genes) < n:
99              raise ValueError(f"Requested {n} genes, but only found {len(genes)} in GFF
                ↪  file.")
100
```

```
        if find_method == "random":
            sampled_genes = genes.sample(n=n, random_state=2024)  # For reproducibility
        elif find_method == "first":
            sampled_genes = genes.head(n)

        upstream_regions = []

        for _, row in sampled_genes.iterrows():
            start = row['start'] - 1  # Convert to 0-based index
            end = row['end']          # End is inclusive in GFF but this is 1-based
            strand = row['strand']

            if strand == '+':
                region_start = max(0, start - upstream_start)
                region_end = max(0, start - upstream_end)
                seq = genome_seq[region_start:region_end+1]
            elif strand == '-':
                region_start = end + upstream_end
                region_end = end + upstream_start
                seq = genome_seq[region_start:region_end+1].reverse_complement()
            else:
                continue  # Skip if strand information is invalid

            upstream_regions.append(seq)

        return upstream_regions
```