Nucleosome Positioning Inference from Cell Free DNA Data

Juozapas Ivanauskas
Gediminas Gaigalas

Supervisor: Erinija Pranckevičienė

January 19, 2025

1 Introduction

1.1 Theory overview

Cell-free DNA is found in circulating plasma, urine, and other bodily fluids and consists of short, double-stranded DNA fragments (typically under 200 base pairs). It mainly originates from the apoptosis of hematopoietic cells, with minor contributions from other tissues. The short half-life of cfDNA suggests continuous release from dying cells and rapid degradation. The cfDNA fragments correspond to nucleosome structures, which may circulate as nucleosomes or chromatosomes. In disease states or physiological conditions, cfDNA can originate from other tissues, and this variation has been used in diagnostics, such as fetal genetic screening and cancer monitoring. However, most diagnostic techniques rely on genetic differences between tissues, which limits their scope.

1.2 Application

A study proposes using cfDNA fragmentation patterns, shaped by nucleosomes, to analyze the origin of tissue without relying on genetic differences, providing broader clinical applications.

One of the fields this method could be applied on is noninvasive cancer monitoring. By analyzing nucleosome spacing patterns and fragmentation in cfDNA, it's possible to infer the tissue of origin of the cancer cells. This is particularly useful in cases where the primary tumor site is unknown or unclear. Also, changes in cfDNA concentration and nucleosome positioning over time can provide insights into tumor growth, response to therapy, and potential relapse. This method is less invasive than traditional biopsies and can be performed more frequently to closely monitor patient status.

1.3 Project's Aim

The aim of this project is to recreate an algorithm for analyzing cell-free DNA (cfDNA), to map nucleosome positions across the genome and identify areas with increased nucleosome protection. By associating these nucleosome locations with specific cell types using DNase-Hypersensitive sites from various cell lines, project seeks to understand how different cells contribute to the cfDNA pool in blood. This research could improve non-invasive methods for diagnosing and monitoring diseases like cancer by studying these DNA markers.

2 Methods

2.1 Data acquisition

Publically available data from the original paper were used. Raw data (fastq files) were downloaded using SRA toolkit (accession id SRR2130050).

The DNase-HS data of various cell lines were downloaded from encode project website. The cell lines that were used: T cells, naive B primary cell, Monocytes-CD14+, Neuron M059J cell line.

2.2 Reads mapping

Paired reads were mapped to genome using bwa mem algorithm. Mapped reads were subsampled for the simplicity of this project, taking 100000000 random reads. Mapping commands can be seen below (GRCh38 genome was used).

```
bwa index <genome.fa>
bwa mem <genome.fa> <read1.fastq> <read2.fastq>
```

2.3 Filtering bam file

Bam files were filtered using samtools, particularly flags 0x4, 0x2, and 0x100. Flag 4 stands for unmapped reads, flag 2 stands for properly paired reads, and 0x100 stands for secondary allignment. In the case sequencing is not paired, flag 2 should not be used.

```
samtools view -b -f 0x2 -F 0x4 -F 0x100 <file.bam> > <filtered_file.bam>
```

2.4 Windowed Protection Score (WPS)

Windowed protection score was computed using custom python script. It is defined for each coordinate as number of reads completelly spanning the window centred at the coordinate - number of reads that have one of their endpoints in this region. Window size used were 120. The full code can be seen in Apendix 4.1.

2.5 Adjusting WPS Scores

WPS scores were adjusted to a running median of zero using 1kb windows and smoothed with Savitzky-Golay filter (window size 21, second-order polynomial). The script can be found in Appendix 4.2.

2.6 Segmentation

The windowed protection score track was segmented into positive value segments, allowing up to five consecutive negative values, and reporting only segments over 50bp. The custom script can be seen in Apendix 4.3.

2.7 Peak Calling

Note: this was not used with our date, because it provided poor results, probably due to low depth of our data used.

For each segment of 150-450 length, peak calling algorithm was applied. This algorith finds maximum sum contiguous regions above median. The region that is found is defined as peak region, and the peak centre is just centre this region. For the peak, a score is defined as maximum value of the peak minus minimum value of the segment. Only peaks of the legth between 50 and 150 bp were reported. The code can be found in the Appendix 4.4.

2.8 Correlation with DHS sites of specific cell lines

DHS sites are where DNase bids, and are expected to have not so strict nucleosome positioning. Hence in the DHS sites distances between our reported regions should be more loose. To identify possible origin of cell free DNA, the mean idstance between reported regions were checked within DHS sites of various cell lines. This analysis were performed using basic command line utilities.

```
bedtools intersect -a <segments.bed> -b <cell_line_DHS.bed> | \
awk -F'\t' '{print ($3+$2)/2}' | \
awk 'NR == 1 {prev = $1} \
NR != 1 {dif = $1 - prev; if (dif > 0 && dif < 500) {print dif}; prev = $1}' | \
awk '{sum += $1} END {print sum / NR}'
```

3 Results

Using methods described above we were able to detect 9963402 segments of increased protection, presumably due to the nucleosomes (see Figure 1). Computing mean spacing of those segments in the DHS sites, we were able to show that in the blood cell lines this spacing is bigger than in

the neural cell line (see Table 1). These finding are consistent with the fact that in the active DHS sites nucleosome spacing should be not so strict and hence spacing should be bigger.

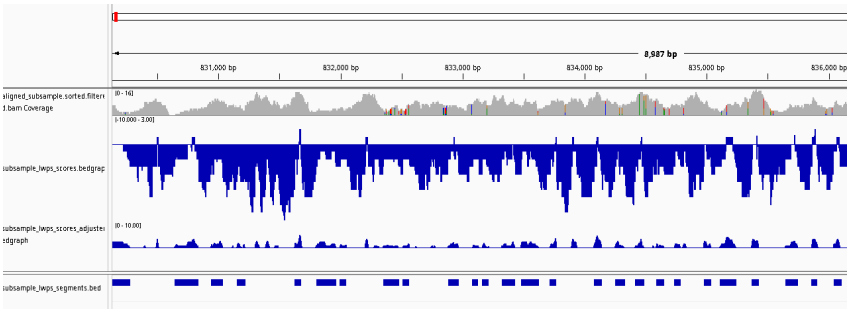

Figure 1: IGV snapshot showing coverage, WPS scores, adjusted WPS scores and segments found.

Cell Type	Mean spacing
T cells	219.937
naive B primary cell	216.996
Monocytes-CD14+	219.531
Neuron M059J cell line	199.923

Table 1: Mean spacing.

4 Appendices

4.1 WPS script

```
import numpy as np
import sys

bed_file_path = sys.argv[1]
output_file_path = sys.argv[2]
window = int(sys.argv[3])
fasta_index = sys.argv[4]
half_window = window // 2

# Getting lengths of chromosomes
chr_lengths = {}
with open(fasta_index, "r") as file:
        for line in file:
                columns = line.split('\t')
                chr = columns[0]
                length = int(columns[1])
                chr_lengths[chr] = length

# Initialise score arrays for each chromosome
scores = { key: np.zeros(chr_lengths[key], dtype = int) for key in chr_lengths }

# Compute wps scores
# Scores are updated read by read, i.e. for each read, affected scores are updated
with open(bed_file_path, "r") as file:
    for line in file:
        # Split the line into columns
        columns = line.strip().split("\t")
        chrom = columns[0]
        start = int(columns[1])
        end = int(columns[2])

        length = end - start
        if length <= window:
            scores[chrom][(start-half_window):(end+half_window)] -= 1
        else:
            scores[chrom][(start-half_window):(start+half_window)] -= 1
            scores[chrom][(start+half_window):(end-half_window)] += 1
            scores[chrom][(end-half_window):(end+half_window)] -= 1


# Write results
with open(output_file_path, "w") as file:
    for key, value_list in scores.items():
        for position, value in enumerate(value_list):
            file.write(f"{key}\t{position}\t{position}\t{value}\n")

# Note, this script deals with entire genome scores at once, which requires a lot of RAM
# If RAM is limited, one should modify script to work chromosome by chromosome.
```

4.2 Score Adjustment Script

```
import numpy as np
import sys
from scipy.signal import savgol_filter
from scipy.ndimage import median_filter
from itertools import groupby

file = sys.argv[1]
output = sys.argv[2]

#  Parameters
window_size = 1000
savitzky_golay_window = 21
savitzky_golay_order = 2

# Function to process scores for a single chromosome
def process_scores(scores):
```

```
    # Adjust to a running median of zero
    median_adjusted = scores - median_filter(scores, size=window_size)
    # Smooth using a Savitzky-Golay filter
    smoothed = savgol_filter(median_adjusted, savitzky_golay_window, savitzky_golay_order)
    return smoothed

def process_file_by_chromosome(input_file, output_file):
    with open(input_file, "r") as infile, open(output_file, "w") as outfile:
        # Take lines from one chromosome
        for chrom, lines in groupby(infile, key=lambda line: line.split("\t")[0]):
            # Read lines for the current chromosome
            chrom_data = [line.strip().split("\t") for line in lines]
            scores = np.array([float(row[3]) for row in chrom_data])

            # Process scores
            adjusted_scores = process_scores(scores)

            # Write updated data to the output file
            for i, row in enumerate(chrom_data):
                row[3] = f"{adjusted_scores[i]:.5f}"
                outfile.write("\t".join(row) + "\n")

process_file_by_chromosome(file, output)
```

4.3 Segmentation script

```
import numpy as np
import sys

scores_file = sys.argv[1]
output_file = sys.argv[2]

current_segment = [-1, -1, -1]
below_zero_count = 0
prev_position = -1

# Iterate through scores, saving positive value segments, allowing up to 5 consecutive negative values
with open(scores_file, "r") as file, open(output_file, "w") as output:
    for line in file:
        row = line.split("\t")
        chrom = row[0]
        position = int(row[1])
        score = float(row[3])

        # If chromosome changes, save current segment (if it is not empty)
        if chrom != current_segment[0] and current_segment[1] != -1:
            current_segment[2] = prev_position
            if current_segment[2] - current_segment[1] > 50:
                output.write("\t".join([str(i) for i in current_segment]) + "\n")
            current_segment = [-1, -1, -1]
            below_zero_count = 0

        # If positive value, start new segment (if not started) and reset below zero count
        if score > 0:
            below_zero_count = 0
            if current_segment[1] == -1:
                current_segment[0] = chrom
                current_segment[1] = position
        # Else update below zero count, if 5 is reached, end current segment
        else:
            if current_segment[1] != -1:
                if below_zero_count == 4:
                    current_segment[2] = position - 4
                    if current_segment[2] - current_segment[1] > 50:
                        output.write("\t".join([str(i) for i in current_segment]) + "\n")
                    current_segment = [-1, -1, -1]
                    below_zero_count = 0
                else:
                    below_zero_count += 1

        # Used in case chromosome switches
        prev_position = position
```

4.4 Peak Calling Script

```
import numpy as np
import sys
from collections import defaultdict

if len(sys.argv) == 1:
    print("Usage: python peakCalling.py <segments> <scores> <output>")
    sys.exit(1)

segments_file = sys.argv[1]
scores_file = sys.argv[2]
output_file = sys.argv[3]


# Function to find maximum-sum contiguous window above the median
def peak(chrom, start, end):
    median = np.median(scores[chrom][start:end])
    max_sum = 0
    current_sum = 0
    peak_start = -1
    peak_end = -1
    tmp_start = start

    for i in range(start, end +1):
        if scores[chrom][i] <= median:
            if current_sum > max_sum:
                max_sum = current_sum
                peak_start = tmp_start
                peak_end = i - 1
            current_sum = 0
            tmp_start = i + 1
        else:
            current_sum += scores[chrom][i]

    if current_sum > max_sum:
        max_sum = current_sum
        peak_start = tmp_start
        peak_end = end

    peak_centre = int((peak_start + peak_end) / 2)
    # Define custom peak score as peak max - region min
    score = max(scores[chrom][(peak_start):(peak_end)]) - min(scores[chrom][start:end])
    return [str(i) for i in [chrom, peak_start, peak_end, peak_centre, score]]


# Load scores
scores = defaultdict(list)  # Dictionary of lists
with open(scores_file, 'r') as file:
    for line in file:
        parts = line.strip().split("\t")
        key = parts[0]
        value = float(parts[3])
        scores[key].append(value)

peaks = []
# Iterate through segments
with open(segments_file, "r") as seg_file:
        chrom, start, end = line.strip().split()
        start, end = int(start), int(end)

        length = end - start
        # Take only segments of required length
        if length > 50 and length <= 450:
            highest_peak = peak(chrom, start, end)
            # peaks can only be in the range 50-150
            if int(highest_peak[2]) - int(highset_peak[1]) > 50 and \
               int(highest_peak[2]) - int(highset_peak[1]) < 150:
                peaks.append(highest_peak)

# Save output
with open(output_file, "w") as output:
    print("Writing output")
```

```
    for peak in peaks:
        output.write("\t".join(peak) + "\n")

# Note: requires a lot of RAM, if not available,
# please modify script to work chromosome by chromosome
```

4.5 Data links

DHS sites:

- T cells https://www.encodeproject.org/files/ENCFF921ZQH/@@download/ENCFF921ZQH.bed.gz
- Naive B primary cells https://www.encodeproject.org/experiments/ENCSR000ELG/
- Monocytes-CD14+ https://www.encodeproject.org/experiments/ENCSR000ELE/
- Neuron M059J cells https://www.encodeproject.org/files/ENCFF290FNL/

Original paper:

- http://dx.doi.org/10.1016/j.cell.2015.11.050