Lab Work 2 Report

Guilherme Amorim, 107162
José Gameiro, 108840
Tomás Victal, 10918

University of Aveiro - TAI

Abstract
In this project, we explore the potential to identify the types of organisms present in a metagenomic sample by comparing their similarity to multiple known reference sequences. To achieve this, we use **Normalized Relative Compression (NRC)** with a **finite-context model**. The implementation follows a methodology that compares one sample against multiple references in a database file. The tool begins by training a finite-context model using the sample, and then computes the NRC value for each sequence of DNA in the database. Based on these values, we rank the top candidate matches the sample.

Contents

1 Introduction

In the context of Algorithmic Information Theory, the compressibility of sequences provides a powerful tool for assessing similarities and identifying patterns in biological datasets. One of the most intriguing challenges in exobiology is the analysis of exogenous metagenomes, which consist of genetic sequences from multiple organisms, often without direct references.

In this practical assignment, our group was tasked to develop a program that implements the **Normalized Relative Compression (NRC)** to estimate which known organisms from a given reference database (provided by the class professors which contains multiple DNA sequences of different organisms) share the highest similarity with the genomes found in a sequenced metagenomic that may come from the European Space Station.

NRC is a measure of similarity or distance between two sequences (or strings) based on how well one sequence can be **compressed using information from the other**. This can be given by the following formula:

$$\mathcal{NRC}(x \parallel y) = \frac{C(x \parallel y)}{|x| \log_2(\mathcal{A})} \tag{1}$$

Where $C(x||y)$ represents the number of bits needed to lossless compress x given exclusively a model trained with y, $|x|$ the size of the sequence x, and $\log_2(\mathcal{A})$ the alphabet of the sequence x, which in this case it will be 4 and, hence, $\log_2(4) = 2$.

To achieve the goals define for this assignment, we followed the following workflow:

1. **Train a finite-context mode (or a Markov Model)** using only the **metagenomic sample** y. During this process, the model will learn the frequency of patterns present in the sample;
2. **Freeze the model's counts**, meaning that after training on y, the model counts will no longer be updated;
3. For each sequence x^i in the reference database of known organisms:
    - **Estimate the amount of bits required to compress** x^i using the model trained on y;
    - **Apply the Normalized Relative Compression (NRC)** to measure the relative compression of x^i to y.
4. **Sort by NRC the names of the sequences** and **print a top** 20.

2 Implementation

The implementation of the finite-context model used for this assignment was the same as the one presented in the last assignment, however a new function was included that will be later explained.

Our implementation is divided into multiple files:

- **data_base_processor.rs**
- **model_saver_loader.rs**
- **metaClass.rs**
- **finite_context_model_image.rs**
- **image_processor.rs**

2.1 Data Base Processor

Following the workflow described in section 1 the first step is to read the **metagenomic sample** that's inside the file ***meta.txt*** and train a finite context model with it. For this we added a new function to our ***file_reader.rs*** called **read_line** to read a specific file line by line. With the sequence obtained, a new model is trained.

The next step is to obtain known DNA sequences that are inside a text file also provided by the class professors (***db.txt***). For this we created a new class called **DataBaseProcessor** with the following methods:

- **new** - Creates a new instance of the **DataBaseProcessor**;
- **get_database** - Retrieves the database containing all the samples;
- **read_samples** - Reads all the samples that are in a text file file. It parses the name of each sample by identifying the character **@** and stores the DNA sequence in a **HashMap**, where the key is the name of the sample and the value is the sequence;

- **get_sequence_by_name** - Retrieves a specific sequence that's inside the database;
- **comparative_nrc_analysis** - Performs a comparative analysis of normalized relative compression (NRC) scores among a list of sequences that have a score bellow a threshold. For each sequence in this list, it constructs a finite context model using the **context size** and a **smoothing factor**, then calculates how well this model compresses every other sequence in the list. Each sequence is compared against the others, and the results are sorted and collected into a vector;
- **export_nrc_comparisons_to_json** - Saves data obtain from the previous function (**comparative_nrc_analysis**) to a json file. To achieve we had to create two structures **ComparisionResult** and **MatchScore**, the first one contains two elements **base_sequence**, the name of the sequence used to train the model and **matches**, a vector with the NRC scores for the other samples , the second one contains also two elements, **target_name**, the name of the sequence that was used to calculate the NRC, and **nrc_score** the NRC score.

2.2 Save and Load a Model

The file ***model_saver_loader.rs*** contains two simple functions:

- **save_model** - Serializes a **FiniteContextModel** instance and saves it in both **JSON** and **BSON**.
- **load_model** - Reads a previously saved **FiniteContextModel** from a **JSON** file, de-serializing its contents and reconstructing the model from the provided file path.

2.3 Meta Class

This file serves as the **main entry point** of the MetaClass pipeline. Its main goal is to **analyze a metagenomic sample** by building a probabilistic model (Finite Context Model) and using it to compare known DNA sequences from a reference database using the **Normalized Relative Compression (NRC)** metric.

It is divided into 6 main parts:

1. **Argument Parsing:**
   - Uses an **argument parser** crate to allow command-line specification of:
     - ***-s***: Path to the metagenomic sample (required);
     - ***-d***: Path to the reference database file (also required);
     - ***-k***: Context size for the finite-context model (the default value is 3);
     - ***-a***: Smoothing factor α (the default value is 0.01);
     - ***-t***: Number of top sequences to display (the default value is 20);
     - ***-l***: Threshold for low NRC scores (the default value is 0.5).
   - Includes validation to ensure parameter ranges are within acceptable bounds.
2. **Model Training**
   - Reads the metagenomic sample character-by-character using a custom file-reader that was explained in the previous report;
   - Trains a **Finite Context Model (FCM)** on the metagenomic sample using the specified ***k*** and α,

recording the time taking for this phase.

3. **NRC Score Computation**
    - Initializes a ***DataBaseProcessor*** with the given reference file;
    - Computes the NRC score for each sequence in the database using the previously trained model;
    - Sorts all sequences by NRC score and displays the top-N most similar ones.
4. **Threshold Filtering and Comparative Analysis**
    - Sequences with NRC scores **below the specified threshold** are considered significantly similar to the metagenomic sample;
    - A **comparative NRC analysis** is conducted on these sequences, and the obtained results are exported to a JSON file.
5. **Complexity Profile Generation**
    - Computes the **complexity profile** of the metagenomic sequence and all sequences with a NRC score below the threshold;
    - Uses an instance of the ***ChartGenereator*** class to visualize these profiles and saves the results as an image.
6. **Performance Logging**
    - Logs the time spent in each major phase:
        - Model training;
        - NRC score computation;
        - Total execution time.

2.4 NRC for Images

2.4.1 Introduction

To further explore the concept of **NRC**, we attempted to apply it in other domains such as images and audio. Due to time constraints and more promising results with images, we decided to focus more deeply on the image domain.

Initially, we used a dataset of grayscale images of human faces. However, the pictures were close-ups, showing only facial features without any context such as head shape or hair. We used an algorithm similar to the one used for DNA sequences, treating the image as a sequence of pixels, and using the last pixels as context the pixels behind the current pixel. This approach did not yield good results, so we began researching alternatives and came across a paper on using **Normalized Compression Distance (NCD)** with **finite context models**[2] and decided to test the same approach but using **NRC**.

2.4.2 Concept

As we explained in the introduction, we initially used the same algorithm for genomic sequences. This created an issue in which the context of each pixel was limited to the k preceding pixels, since the image was treated as a one-dimensional sequence(Figure 1).

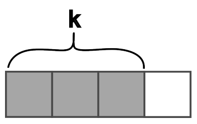

Figure 1: Context of a pixel with any k for first implementation.

This proves to be a limitation when working with images, as they are represented in a 2D space, not as a simple sequence. This means we can access context around a pixel. However, since we are using compression, we can only use as context pixels that are already uncompressed, specifically, the ones that appear in the rows above and in the same row in columns preceding the pixel. To solve this, we started implementing the context presented in the paper[2], but only for **k**=2, where the context pixels are, one directly above and another one directly behind the current pixel, as shown in the image(Figure 2).

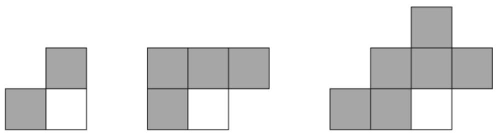

Figure 2: Context templates used by the encoder, corresponding to model orders of 2, 4 and 6.[2]

This significantly improved the results; however, it was still somewhat distant from the results presented in the paper. Therefore, we decided to fully implement the method used in the paper, where the **NRC** is calculated by a weighted average of three models with k=2,4,6. The context representation for this method can be seen in Figure 2. According to the following expressions from the paper[2]:

$$P(X_{n+1} = s) = \sum_{k=2,4,6} P(X_{n+1} = s \mid c_{k,n}) w_{k,n}$$

$$w_{k,n} \propto w_{k,n-1}^{\gamma} P(X_n = x_n \mid c_{k,n-1})$$

$$\sum_{k=2,4,6} w_{k,n} = 1.$$

2.4.3 Implementation

2.4.3.1 Finite Context Model for Images

The finite-context model class was modified to comply with the image requirements. The functions that were altered or added are as follows:

- **train_mat_image** - changed to use a `Mat` from OpenCV instead of a sequence.
- **get_context** - a function that calculates the context based on the current pixel, the frame, and the desired k, where $k \in [2, 4, 6]$. It uses an auxiliary function **is_in_frame** to check if the context pixels are within the frame. If any pixel is out of bounds, the context pixel is replaced by the current pixel.
- **compute_probability** - modified to support both the pixel and the context of the image.
- **calculate_information_content** - updated to use a `Mat` and iterate over all pixels of the image.

2.4.3.2 Image Processor

This class is responsible for loading the dataset, processing quantization, and computing the NRC for the saved images with the given models:

- **compute_nrc_multi_model** – computes the NRC for multiple models, all using equal weights.
- **compute_weighted_nrc** – computes the NRC using weighted average based on the behavior of each model.
- **compute_nrc** – computes the NRC for a single model.
- **quantize_image** – quantizes the image using the Lloyd-Max quantizer.

2.4.3.3 Image

This is the main class to process a sample image and an entire database full of them, the flow of the class processing is the following:

- Parse command-line arguments using `argparse`, including:
  - **-i**: path to the input image file (required)
  - **-d**: path to the database file (required)
  - **-a**: smoothing parameter α (default: 0.5)
  - **-t**: number of top sequences to display (default: 10)

  - **-l**: number of quantization levels for the image (default: 256)

- Load the dataset using the **ImageProcessor** class with the database path provided.
- Read the input image in grayscale using OpenCV and apply quantization.
- For each k, create a new **FiniteContextModelImage** with the smoothing parameter α, train it on the quantized image, and store it in the map.
- Compute the weighted NRC score for each image in the dataset using the trained models and a decay factor $\gamma = 0.99$.

3 Experimental Analysis

In this section we present a comparative analysis to evaluate our implementation, in terms of performance, accuracy, etc. For this we developed multiple experiments like, see the affect of k and alpha values, compare complexity profiles between the samples that have a lower NRC score, evaluate the accuracy, apply our new finite context model for images and see the NRC results.

3.1 Top 20 sequences contained in Metagenomic Sample

In the first experiment, we aimed to identify which known genomic sequences are most similar to the metagenomic sample provided. To do this, we trained a Finite Context Model using the full metagenomic data and evaluated all sequences from the reference database using the **Normalized Relative Compression (NRC)** metric. The experiment was conducted with $\mathbf{k = 10}$ (context size) and $\alpha = \mathbf{0.01}$ (smoothing factor), as these parameters provided a good balance between context sensitivity and generalization.

Table 1: Top 20 Sequences ordered by NRC score

Rank	Sample Name	NRC
1	@NC_001348.1 Human herpesvirus 3, complete genome	0.196636
2	@NC_005831.2 Human Coronavirus NL63, complete genome	0.241053
3	@Super ISS Si1240	0.262869
4	@OR353425.1 Octopus vulgaris mitochondrion, complete genome	0.310705
5	@Super MUL 720	0.397340
6	@gi|49169782|ref|NC_005831.2 |Human Coronavirus NL63, complete genome	0.735003
7	@gi|14141972|ref |NC_002786.1 |Maize rayado fino virus isolate Costa Rican, complete genome	1.507647
8	@gi |971748077 |ref |NC_028836.1 |Pseudomonas phage DL62, complete genome	1.510804
9	@gi |9629255 |ref |NC_001793.1 |Oat blue dwarf virus complete genome	1.517540
10	@gi |84662653 |ref |NC_007710.1 |Xanthomonas oryzae phage OP2 DNA, complete genome	1.522362

11	@gi |472339725 |ref |NC_020840.1 |Tetraselmis viridis virus S20 genomic sequence	1.547256
12	@gi |475994049 |ref |NC_020902.1 |Equine Pegivirus 1 isolate C0035, complete genome	1.555544
13	@gi |927594140 |ref |NC_027867.1 |Mollivirus sibericum isolate P1084-T, complete genome	1.555690
14	@gi |971752197 |ref |NC_028885.1 |Pseudomonas phage DL64, complete genome	1.556349
15	@gi |22855189 |ref |NC_004168.1 |Yellowtail ascites virus segment A, complete sequence	1.559760
16	@gi |33867953 |ref |NC_005075.1 |Helicobasidium mompa No.17 dsRNA virus small segment, complete sequence	1.560387
17	@gi |113478394 |ref |NC_008311.1 |Norovirus GV, complete genome	1.567778
18	@gi |50253405 |ref |NC_005997.1 |St Croix river virus segment 1 putative RNA-dependent RNA polymerase VP1 gene, complete cds	1.573770
19	@gi |21427658 |ref |NC_004018.1 |Gremmeniella abietina RNA virus MS1 RNA 1, complete genome	1.574988
20	@gi |33867950 |ref |NC_005074.1 |Helicobasidium mompa No.17 dsRNA virus large segment, complete sequence	1.577041

A few key observations can be drawn from these results:

- **Human-associated viruses rank among the top**, including *Human herpesvirus 3* (NRC = 0.1966) and *Human Coronavirus NL63* (NRC = 0.2411) These low values suggest a high degree of similarity with the patterns present in the metagenomic data, potentially indicating contamination, cohabitation, or shared genomic characteristics;
- Two entries labeled **@Super ISS Si1240** and **@Super MUL 720** also appear in the top 5. These might be synthetic constructs, assemblies, or sequences derived from previous space, related studies. Their presence hints at content in the metagenomic sample that aligns closely with known ISS-associated sequences;
- Starting from rank 6, the NRC values increase sharply. The duplication of the *Human Coronavirus NL63* entry in a different identifier format (rank 6) reinforces the idea of consistent similarity detection across naming schemes and identifiers;
- Sequences with **NRC values above 1.5**, such as *Maize rayado fino virus*, **Pseudomonas phage**, and *Xanthomonas oryzae phage*, show considerably lower similarity. These results may reflect incidental or random matches due to limited pattern overlap;
- The **diversity** of organisms present in the top 20 includes phages, plant viruses, mollivirus, norovirus, and even RNA viruses. This diversity suggests the metagenomic sample could be a complex mixture of genetic material from different sources — either real, synthetic, or environmental.

3.2 Experimenting with different values of k and alpha

To better understand the impact of the model parameters on the NRC computation, we designed a second experiment focusing on the influence of the **context size *k*** and the

smoothing factor α. This experiment was divided into two parts:

- **In the first part**, we varied the value of $\boldsymbol{k}$ from 2 to 20 in increments of 2, while keeping α fixed at **0.01**. This allowed to observe how increasing the context window size affects the model's ability to capture patterns and perform compression

- **In the second part**, we fixed the value of $\boldsymbol{k} = \boldsymbol{10}$ and varied α over a range of values: **0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5**. These values were chosen to explore both very low and relatively high smoothing effects on the model's behavior.

The results are presented in the following figures:

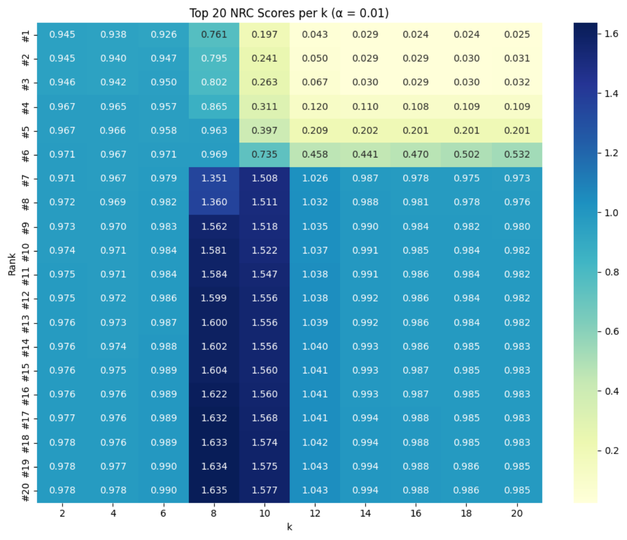

Figure 3: Results obtained by altering the values of k

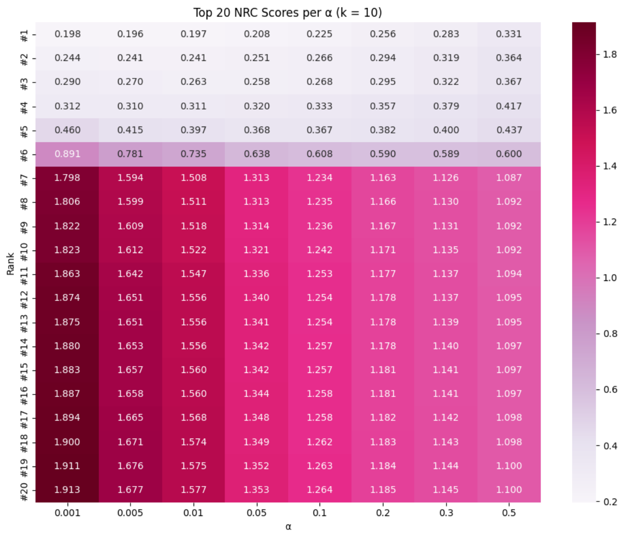

Figure 4: Results obtained by altering the values of alpha

From the first heatmap Figure 3, we observe that the NRC scores tend to decrease as the value of k grows, especially from k = 8 onward. This trend is particularly evident from rank 6 to 20, where the scores stabilize and reach higher values with increasing k. In contrast, the top five ranks show relatively lower scores at higher k, with noticeable spikes at lower k values (especially k = 8 for ranks 6–20). This suggests that smaller k values may provide lower sensitivity for identifying top-ranked sequences.

In the second heatmap Figure 4, increasing the α value results in a general increase in NRC scores, especially for ranks 6 through 20. The scores for the top five ranks remain relatively low across all α values, with only gradual increases. However, for lower-ranked sequences (e.g., 6–0), the effect of α is much more pronounced, showing a steady rise in NRC values with higher α. This indicates that higher α values may help in amplifying the contribution of weaker signals, thereby enhancing the detection of less dominant sequences in the dataset.

3.3 Execution Time

In this experiment, we analyzed the computational cost of our method by measuring the

training time, **NRC computation time**, and **total execution time** across multiple combinations of k (ranging from 1 to 20) and three values of α (0.01, 0.1 and 1). For each pair (k, α). For this we compiled our program using the flag *–**release*** and obtained the following results:

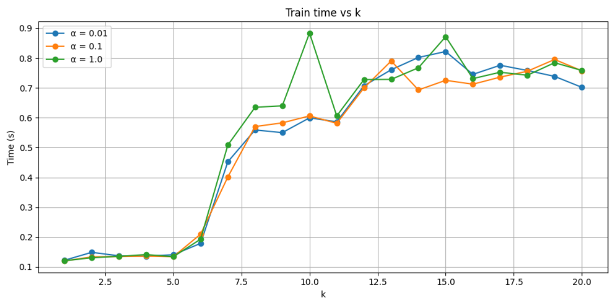

Figure 5: Training time variation

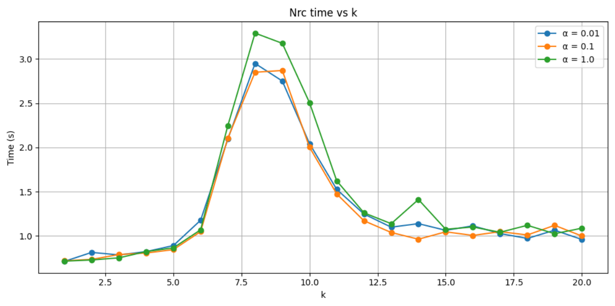

Figure 6: NRC computation time variation

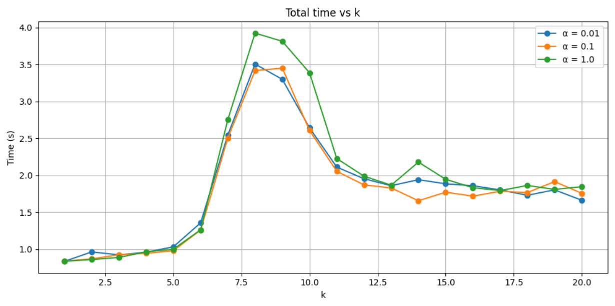

Figure 7: Total execution time variation

By observing this visualizations it is possible to conclude that:

- In Figure 5, the time taken to train a model **increases** with **k** and does **not peak sharply** like NRC time. Training is generally **less sensitive to** α, but with $\alpha = 1.0$ **is again slower**, especially at mid to high ***k***. From $k = 10$ to $k020$, the training time is more stable but still slightly increasing.
- In Figure 6, for all α values, NRC time rises steeply until $k = 8$, then drops off. $\alpha = 1.0$ shows the highest peak, followed by 0.01 and 0.1. This may indicate that **higher** α **values slow down the NRC process** when k is small to mid-range. After $k = 10$, times converge and stabilize near 1s, with minor fluctuations
- In Figure 7, total time closely follow the NRC time trend, which implies NRC is the **dominant contributor** to overall time. The difference between α values becomes more noticeable at peak (again, $\alpha = 1.0$ is slowest), but less so at higher k. After $k = 10$, all α values result in more consistent and lower total times.

With all of this we can conclude that **mid range k values (around 7-9)** should be **avoided**. **Use** α **within a range of 0.01 and 0.1** for better performance and **higher k values (¿ 10)** seems to bring more consistent and manageable times for all phases.

3.4 Accuracy of the solution

One of the most important experiments we conducted was to evaluate the **accuracy** of our solution—specifically, its ability to assign **lower NRC scores** to sequences that are truly part of a metagenomic sample.

To perform this experiment, we used the tool GTO to generate random sequences. These sequences allowed us to build a new test database and simulate different mutation levels on sequences embedded within metagenomic samples.

We generated a test database containing **50 random samples**. The length of each sequence was randomly chosen between **1,000 and 10,000** symbols. For reproducibility, the sample identifier served as the seed for each sequence generation.

With the test database ready, we proceeded to construct **7 metagenomic samples**. In each of these, **5 random sequences** from the test database were included. These same 5 sequences appeared in all 7 metagenomic samples but with **varying mutation rates: 0%, 1%, 5%, 10%, 15%, 20%, and 25%** respectively.

To introduce additional randomness, each metagenomic sample was also supplemented with **150 new random sequences, 30 per each original sequence included in the metagenomic sample**. These were generated by randomly selecting sequences from the database and creating a new **151-base-long subsequence** using each sequence that was randomly selected. These random sequences were all included in each metagenomic sample.

The sequences which suffered mutation were **@seq_48, @_43, @seq_19, @seq_14, and @seq_23**. The values of k and alpha were set to 10 and 0.01, respectively.

Now with the metagenomic files and the test database created, we created a simple ***Python*** script, that indicates the top 20 sequences that are present in the metagenomic sample for each sample created, And obtained the following results:

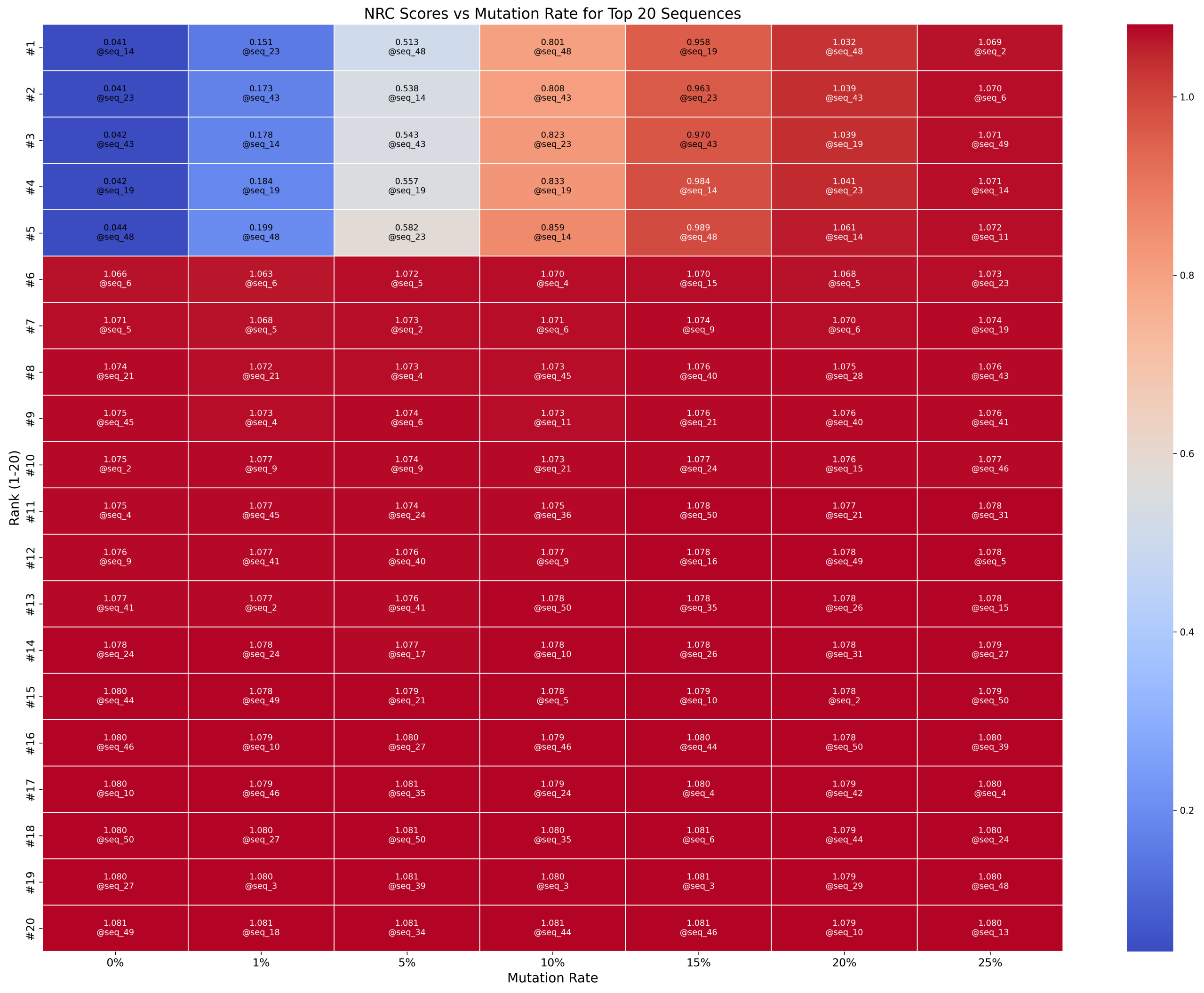

Figure 8: Results for the accuracy experiment

Figure 8 demonstrates that:

- The sequences known to be part of the metagenomic samples (even with increasing mutation rates) generally exhibit **lower NRC scores** compared to the background random sequences;
- The heatmap shows that the five target sequences (**@seq_48, @_43, @seq_19, @seq_14, and @seq_23**) consistently appear within the **top 20 ranked sequences** across all of the mutation rates;
- As the mutation rate increases from 0% to 25%, there is a **noticeable trend of increasing NRC scores** for the target sequences. This is expected, as higher mutation levels make the sequence more dissimilar to the original database sequences;
- As expected, at 0% mutation, the original sequences likely have the lowest NRC scores and are ranked very high.

3.5 Complexity Profiles of the top sequences

Another experiment that we conducted was to evaluate the complexity profile of sequences that had a NRC score below a threshold defined previously.

For this a new function was added to the ***finite_context_model.rs*** to compute the complexity profile for a given sequence, based in a model trained with a metagenomic sequence, called ***complexity_profile***. It takes a string as input and calculates the complexity profile as a vector of floating-point numbers. For each character in the text (considering a sliding window of size $\mathbf{k+1}$), it computes the **bit cost** of predicting that character based on the preceding **k** characters (the context). This bit cost is derived from the logarithmic probability of the character appearing given its context, as learned by the model.

Some other functions where added ***chart_generator.rs***:

- ***group_by_identifier***: Organizes a list of names and their associated data into groups based on a common identifier found within the names;
- ***extract_identifier***: Takes a name os a sequence and tries to pull out the sequence identifier from it using pattern matching. If no specific pattern is found, it tries to get the first word of the sequence name;
- ***smooth_profile***: Takes a list of numbers and calculates a smoothed version of it by averaging values within a sliding window. This helps to reduce noise in the complexity profile of a sequence.
- ***draw_complexity_profiles***: Represents the complexity profiles calculated previously in a line chart. If multiple sequences have the same identifier, then the complexity profile will appear in the same chart window with different colors. It also applies smoothing to the complexity profiles by using the ***smooth_profile*** function.

For this experiment the threshold was set to 0.8, the model was trained with 10 and 0.01 for k and alpha, respectively. The results can be seen in the following figure:

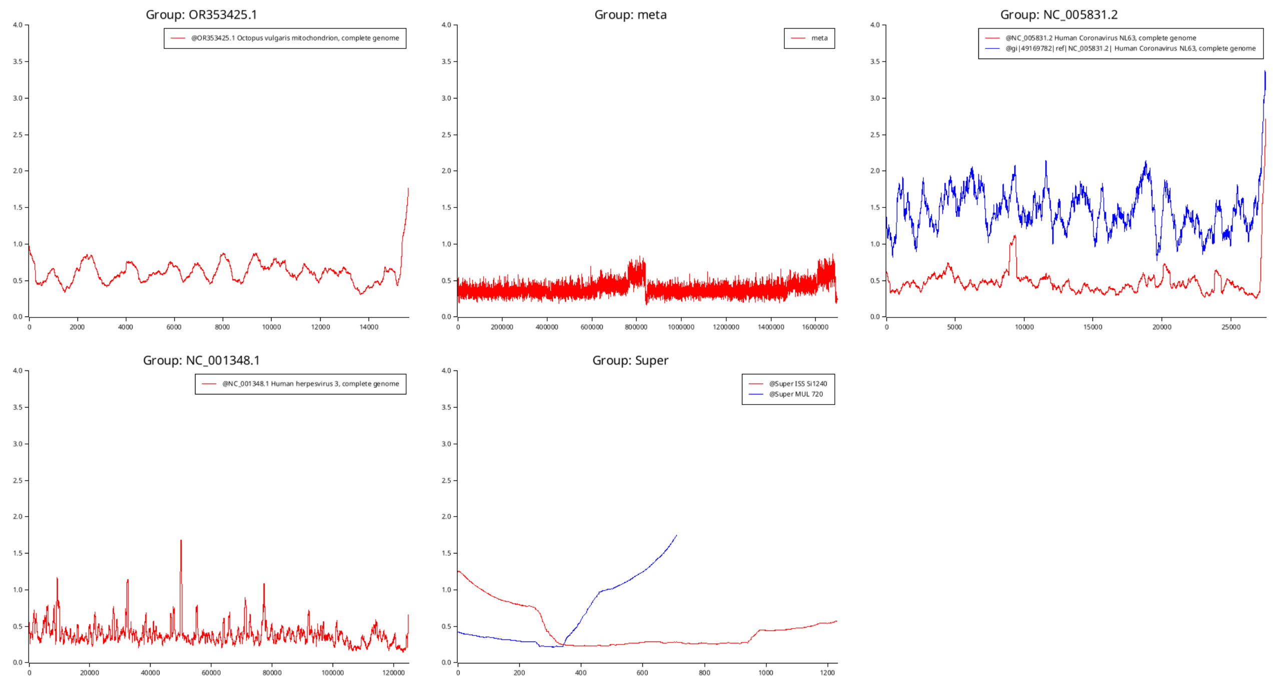

Figure 9: Complexity profiles of sequences with a NRC score lower than 0.8

By observing Figure 9 we can conclude that:

- When visualizing the **Group NC_005831.2**, we can see that the sample with the red line is more similar to the meta one, when compared with the blue line. This means that the the sample represented with the red line has a higher probability of being present in the metagenomic sequence provided by the class professors;
- For the **Group: Super** we can see that the blue line has more similarity with the metagenomic sequence at the beginning, and the red line is more similar after colliding with the blue line. This could indicate that both of these samples are present in the metagenomic sample.

3.6 Compare NRC Scores between top sequences

Another experiment that we performed was to evaluate the similarity between all sequences that have a NRC score below a custom threshold.

For this two new functions were added to the ***data_base_processor.rs*** file called ***comparative_nrc_analysis*** and ***export_nrc_comparisons_to_json*** that were previously explain in subsection 2.1.

A finite context model was trained with 10 and 0.01 for the values of k and alpha respectively, and the threshold was set to 0.8.

The results can be seen in the following figure:

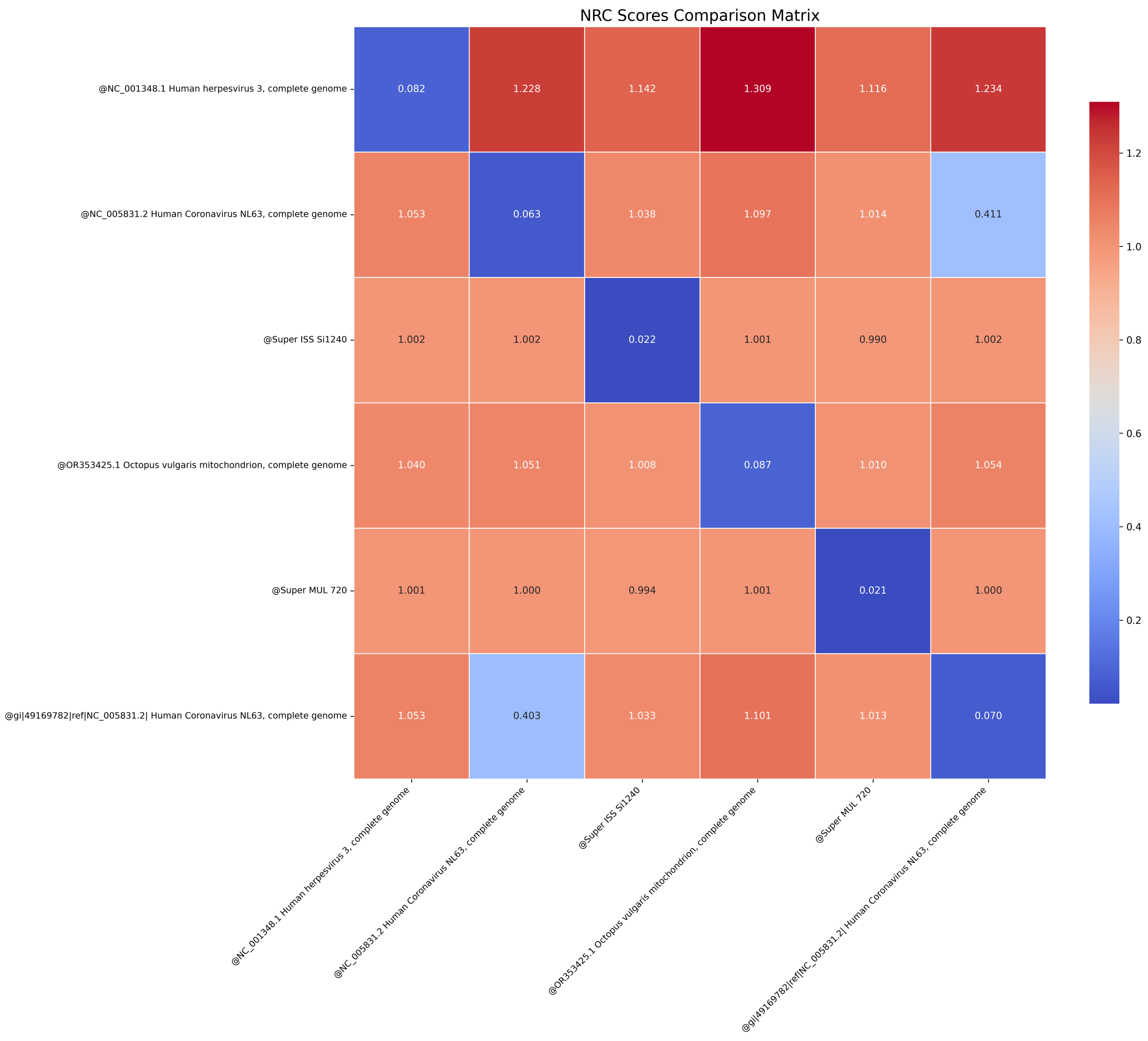

Figure 10: Comparison of NRC scores between top sequences

- Each base sequence has its **lowest NRC score when compared to itself**, as expected (scores close to 0);

- **Sequences from the same or similar organisms** (like the tow *Human Coronavirus NL63 samples*) **show relatively low mutual NRC scores**, indicating high similarity;

- Comparisons with unrelated sequences consistently yielded **higher NRC scores (1.0 or above)**

3.7 Image NRC results

The dataset used was the Olivetti Research Laboratory (ORL) face dataset[1]. It contains 400 images of 40 subjects, with 10 images per subject. To test our solution, we used the first image of each subject and compared it to all other images in the dataset. The goal was to identify the best results, expecting that the top 10 **NRC** values for each image would contain 10 images of the corresponding person.

The table shows reasonable results, but they are not as promising as those reported in the paper. This discrepancy may be due to the fact that the **NRC** performs worse than the Normalized Compression Distance (NCD) when comparing images.

Table 2: Image Count per Person

Person	**# Images**
1	6
2	10
3	4
4	5
5	6
6	3
7	7
8	8
9	3
10	3
11	6
12	5
13	6
14	6
15	1
16	1
17	2
18	6
19	8
20	3
21	4
22	8
23	4
24	6
25	5
26	3
27	8
28	3
29	5
30	7
31	3
32	1
33	2
34	8
35	2
36	2
37	5
38	5
39	1
40	1
41	10
Sum	**192**

3.8 Image Quantization

From the tests we conducted, changing the quantization levels has a significant impact on the NRC algorithm, making its performance worse when the quantization levels are low. For example, in the image below (Figure 11), we can see that the best results occur when there is no quantization or when the quantization level is set to 256.

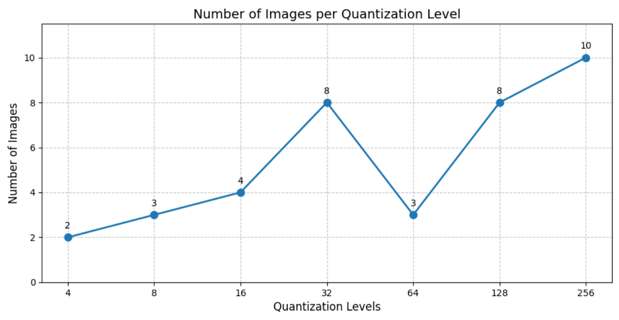

Figure 11: Results for different quantization levels

4 Instructions

How to install dependencies, compile, and run the programs

4.1 Dependencies

Before running the programs, *Rust*, *Cargo* and **OpenCV** need to be installed. Additionally we used the following packages from rust:

- ***argparse***, version **0.2.2**
- ***plotters***, version **0.3.7**
- ***rand***, version **0.9.0**
- ***serde***, version **1.0** and an additional feature called **derive**
- ***serde_json***, version **1.0.139**
- ***bson***, version **2.0**
- ***opencv***, version **0.94.4**
- ***regex***, version **1.11.1**

To install Rust and Cargo, run the following command (if on Linux or macOS systems):

```
curl https://sh.rustup.rs -sSf | sh
```

On Windows download and run this file rust-up-init.exe.

To install OpenCV, follow the tutorial through this link.

4.2 Compiling the Project

To compile the project, navigate to the root directory and run:

```
cargo build
```

After compilation, executables will be available in the 'target/debug' folder.

4.3 Running Meta Class

To execute the Meta Class executable run:

```
target/debug/metaClass
-s {metagenomic_sample_path}
-d {db_path}
-k {k_value}
-a {alpha_value}
-t {top_sequences_display}
-l {threshold}
```

Where:

- **metagenomic_sample_path**: Path to the text file that contains the metagenomic sample;
- **db_path**: Path to the text file that contains a database with known samples;
- **k**: Context size, i.e., the number of characters considered before the current character;

- **alpha**: Smoothing parameter to avoid zero probabilities;
- **top_sequences_display**: Number of sequences to be displayed;
- **threshold**: A value between 0 and 1.0 for the experimental analysis.

4.4 Examples

A predefined bash script is available in the 'scripts/bash' directory to execute the Meta Class. To execute the ***generate_meta.sh*** it is required to have gto installed. To do so follow this tutorial

To execute the bash script for the Meta Class follow the following commands:

```
chmod u+x ./run_meta.sh
./run_meta.sh
```

Also some python scripts are also provided to generate visualizations, they can be found in the 'scripts/python' directory. All the visualizations created are saved to the 'visualizations' directory. To execute the python files its required to first create a virtual environment if none exists

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
python3 <name of the file to execute>
```

4.5 Running the image class

First you need to download a dataset, like the ORL[1] the dataset used in the assignment.

To execute the class run:

```
target/debug/image
  -i {image_to_compare}
  -d {path_folder_with_images}
  -a {alpha_value}
  -t {top_images_display}
  -l {quantization}
```

One example of execution would be:

```
target/debug/image
    -i data/images/1_1.jpg
    -d data/images
```

5 Conclusions

The goal of project was to develop a system capable of identifying the most similar organisms to a given metagenomic sample using NRC, built upon Finite Context Models. Throughout our experiments, we observed the impact of model parameters on the system's performance:

- Increasing k improves the model's ability to capture sequence structure, leading to more precise matches, but at the cost of increased computation time.
- Lower alpha values tends to improve specificity in data. In contrast, higher alpha improve robustness, i.e, the results are more generics.

Furthermore, we also implemented a feature that applies NRC concepts to image data, using the approach for face detection. With this approach, we obtained some interesting values, although they were not the best. However, we were able to understand how this could be implemented and the theory that makes it possible, even using NCD.

This project demonstrates the potential of algorithmic information theory as a practical tool for biological data analysis and we can conclude that with the integration of NRC and FCMs, it is possible to build models capable of identifying similarities in metagenomic data, even in the presence of mutations, noise, or unknown sequences.

References

[1] marlon tavares. The olivetti research laboratory (orl) face dataset. https://www.kaggle.com/datasets/tavarez/the-orl-database-for-training-and-testing.

[2] Armando J. Pinho and Paulo J. S. G. Ferreira. Image similarity using the normalized compression distance based on finite context models. https://ieeexplore.ieee.org/document/6115866.