MATSMATS: Matrix-Sequence Matching & Scoring

Jack Fraser-Govil

July 18, 2025

1 Background

The goal of the project is to improve a deterministic/theoretical model by comparing its output to the better-performing models submitted to the IBIS Challenge. Though these models outperformed the theoretical model on some counts, many of them are 'Black Boxes' which do not necessarily provide biological insights; despite performing better on benchmarks.

2 Problem Statement

The theoretical model performed notably better in the PWM-prediction portion of the competition than in the AAA portion; despite the score (nominally) being simply computable from the PWM and the sequences. Ilya suspects this is due to a computational limit encountered.

If $\{S_0, ..., S_{N-1}\}$ are the N sequences in the training data, and $\{W_0, ..., W_M\}$ the M predicted PWM matrices, then to extract the AAA score nominally requires:

- Iterating over all sequences
- For each sequence, iterate over all matrices
- For each sequence-matrix pair, compute the maximally-scoring subregion ($\pm$ a normalisation)

If we assume that each sequence is of length ℓ_s and each PWM encodes ℓ_m bases, then this requires computing $\approx N \times M \times (\ell_s - \ell_m)$ scores, each of which requires ℓ_m individual products.

The code Ilya is using writes every score above a certain threshold to disk - if the threshold is set too high, then no score is output. If the score is set too low, then every score is written to disk.

A cursory look at the data tells me that $N \sim 10^4$, $M \sim 10^2$ and $\ell_s \sim 10^2$ and $\ell_m \sim 10$. If recording each score requires ~ 6 characters, then this translates to $\approx$ 20GB per assay-type. The bottleneck is writing all of this to disk. Ilya therefore resorted to not iterating over every matrix, and randomly sampling in the hope of stochastically capturing the best one.

However, for the problem at hand - writing every single score to disk is highly undesirable. Ilya wants the single, maximal score output: one score per sequence (and the matrix to which it best-matched).

3 Solution

As with most computationally-bottlenecked code, I think the solution is to switch to using C++, and try to do as much in-memory as possible.

I'm calling this **Mat**rix-**S**equence **Mat**ching & **S**corer (MATSMATS), because I like giving things silly but pronounceable names.

The total size of the input data is $N\ell_s + 4W\ell_m$ which should trivially fit in memory (or which can trivially have data piped in).

As part of the RAMICES III & GenCHORD code I already have:

1. A highly efficient and thread-safe input parsing code
2. A well documented argument handler
3. A parallel job scheduler
4. An efficient matrix scanning algorithm

This problem should be almost trivially solved by simply stapling those elements together.

The only point of difficulty will be error-handling, ensuring consistency with prior scoring systems and ensuring that the input data is in the expected format.

3.1 Specification

We will therefore write a code which:

1. Accepts as input:
   (a) A path to a file containing the target PWMs in a standard format
   (b) A path to a file containing the target sequences **OR** target sequences passed via an input pipe stream
2. Outputs a single file containing for each input sequence
   (a) The sequence
   (b) The index of the best-matching matrix
   (c) The score associated with that matrix

This code will (probably) be able to run in parallel for additional computational gains.

3.2 Additional Thoughts

It would be trivial, i.e. to extend the code to include the scores of all matrices (not just the best scoring ones) for comparison. This would increase the output filesize by $\sim M$, but may be more informative.

We could also specify a threshold below which a sequence is determined to have no matches in the input matrices – i.e. flag them as those which have not been adequately captured by any of the PWMs.

It would also be easy (in terms of writing the code, not in terms of the computational cost) to account for short insertions and deletions in the matching process, if that's something it would be interesting to examine.