Impact of minimap2 parametrization on mapping rate and functional Annotations of Metagenomics samples

# Introduction

Metagenomics samples are characterised by a high biodiversity and low purification. Metagenomics samples are taken to answer two main questions:

1. Sample composition: abundance of species in the sample
2. Sample function: what are the main tasks conducted by the species present in the sample

Answering both questions is approached by conducting sequencing experiments and downstream analysis. This project focuses solely on the second task. In sequencing workflow the experiment is followed by the steps base calling and mapping. Based on obtained mappings functional annotation can be conducted. There are several possible methods of approaching functional annotation. One of this options is querying pathway databases, such as KEGG. Especially for metagenomics samples, that contain less studied organisms these yields only few hits. Therefore, we are focusing on GO functional annotations and EC numbers. Both annotation types can be found in genbank files. We aim to find all annotations available for our mapping results.

Next to the characteristic of fewer annotations for less known species, nanopore sequencing of metagenomics samples itself is challenging already. Base calling models are trained on purified human samples, which might cause troubles in accuracy of reads, which consequently, impacts the mapping step. We aim to experiment with parameter settings of the mapping tools to see whether the described problems can be approached by different parameter settings. We want to find parameter settings, that change the mapping behaviour from prioritzing alignment scores to prioritizing better known genomes if possible. This approach should definetly help to increase the number of found annotations but might as well help to overcome the problems arising from lower accuracy. Penalizing low alignment scores less allows mapping of noisier reads.

This project yields big data and therefore requires specific methods to handle. Sequencing results (base calling already conducted) contain couple of 100.000 reads, which need to be mapped against the collection of all known (491.594) bacteria genomes [bacterial genomes]. The mapping step results in a file with mapping information for each read, regardless whether or whether not the read was mapped. For each original read the genome + annotation (genbank file) needs to be downloaded and analysed for the genes spanned by a read and the annotations for these genes. Sequential processing would be time consuming. Furthermore, the reads information is independent from each other meaning, several reads could be processed at the same time without loosing any kind of information.

Material

A read file was provided in fastq format. The reference genomes were available as fna format or as pre-index mmi file. The mapping algorithm used by minimap2 is based on the seed and extend approach, this parts the algorithm in two main parts: indexing the reference data conduct seed and extend alignment for the reads onto the indexed references. Using a pre-indexed files saves time, which is the reason I used the .mmi file.

Methods

The mapping of the reads onto the pre-indexed references sequences was executed in parallel fashion by exploiting minimap2’s multithreading option. The process was split up into 16 processes and scheduled as SLURM task. The mappings were produced with nanopore sequencing standard parametrization (run 1: -ax map-ont -t 15) and an alternative parametrization (run 2: -ax map-ont -t 15 -k 12 -B 2) for comparison purpose.

Annotations were extracted, processing the mapping files using server client model setup using pythons multiprocessing library together with queue objects. At first, one server and 4 clients with 4 peons each were used. For the final two runs, the mapping files where sectioned into 4 parts per file. Each run was conducted with 4 servers, from which each server processed one partition of the mapping file by instructing 4 clients with 4 peons each. In total 8 server with 32 clients and 128 peons were used to process the two mapping files (4 server, 16 clients, 64 peons for each file). From the mapping file gi identifier was extracted to obtain the genbank files. Start position of the read mapping together with the length of either the raw read sequence or if missing length of the clipped cigar[[1]](#footnote-1) string else 1 were used to get the stop position of a mapped read. Genbank files were scanned for the read start – stop range, to obtain annotations for spanned reads. The results (Annotations and their counts) were appended to a temporary file by each peon after completing a single job (one read). After completing a run, all temporary results were collected combined and written to final result files and final visualisation plots were produced and written to file.

Results

To find optimal parameter settings for metagenomics samples two different parametrizations of minimap2 were used. The mapping rate for standard parametrization (run 1) and an alternative (run 2) are shown in table 1, indicating xx.

Histograms of found GO functions

Technical Discussion

Minimap2 offers plenty parameter tuning option as well as output format options. I have received all reads processed by the tool. It would have been possible to only save mapped reads (parameter: --sam-hit-only), which would have decreased the file size tremendously, concesquently resulting in less tasks for the Annotation extraction. For future usage, excluding unmapped reads could speed up the annotation routine.

For the annotation extraction routine I have considered two different set ups to conduct the task. One option would have been scheduling not only the mapping step but as well the extracting of annotations using SLURM. In this case SLURM would have received the server and clients with pre-defined workload per client as jobs. I have decided against this approach, due to the heterogenous runtime per task. In my approach a reference file is downloaded only if it is not present in a chache already. Furthermore, not every read was mapped. In both cases the processing time is significantly lower. For that reason I decided to keep the system dynamic, so that each peon can pick up a new job when completed the current job. I have kept every job to be exactly one read. Variation in job size by partitioning the mapping file looping per peon through the assigned lines conducting the same instructions as in the one line apporach would have been possible (function was developed see test folder). However, the one line job takes already quite some time, so the reporting back in one line packages does not lead to a bottleneck. Processing the mapping file as a whole took a lot of time, therefore I split the mapping file into several sections, which decreased the runtime per run from one week to two days.

At first, results were collected by the server and written to file once the run got completeed. This approach is risky for several reasons:

1. if something goes wrong and the run gets terminated or stuck, for which the chances are increasing with increasing size of data, all data that has already been processed is lost.
2. It is difficult to meet the termination criteria for the server to shutdown clients and itself.

For this routine there are several things that could go wrong such as invalid formats within the mapping file, missing genbank files or malformatted genbank files. Therefore, optimizing the system towards robustness should be the primary goal. I have done this by writing every result immediately to a temporary file maintained by the peon as well as being overly generous with exceptions, to minimize the risk of a job failing and causing problems with meeting termination criteria. For the future I should have considered, saving the results on the server side as well. Furthermore, I should have stored my logging elements instead of just printing them. This would allow to find failed jobs which could be fed in a “repair-run” to the system.

Research Discussion

The chosen method leads to an undercount of annotations in cases where neither the read sequence nor the cigar string is available. In these cases the read length is set to one. For these exceptional cases, the genes that would be spanned by the read are not considered, only the gene the read is located at. Inversely, an overcount arises in cases, where a read does not span a gene completely, regardless all annotations of the gene are counted. The assumption was, that such cases would be rare and therefore negligble. Since I have not collected data on the frequency of such cases it remains unclear if the assumption is correct.

// maybe refer to metagenomics studies that use standard parameters, such as: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-018-5094-y#ref-CR16

References

Bacterial genomes: https://www.ncbi.nlm.nih.gov/genome/browse#!/prokaryotes/

Appendix 1

The cigar string encodes the mapping of a read in that sense, that each base is represented with encoding M for match/mismatch, I for insertion, D for deletion and S or H for soft or hard clipping. Extended cigar string format can be obtained, then M is replaced by X for mismatches and = for matches. The cigar string is a more condensed representation than the actual read sequence since neighbouring bases with the same encoding are summed and the sum is added to the cigar string as the prefixes for the encoding they refer to. Clipping is applied to the ends of the reads to increase the alignment score. The start position given for a mapped read is the first not clipped base of a read. There are R Bioconductor packages available to extract information from cigar strings. My own python script is reduced in functionality to calculate the read length.

1. More on cigar string in appendix 1 [↑](#footnote-ref-1)