Real time analysis of reads from Nanopore Sequencing for fast identification

Genorobotics EPFL semester project - Vladimir Hanin

Abstract— **Extracting the DNA of a plant can be an effective way to identify the species of the plant of interest. Such a method requires the use of a DNA sequencer that outputs a lot of data over time. In this report, a real time pipeline is proposed to interpret the data as it is coming out of the sequencer. This enables to obtain a result must faster, and also enables the stopping of the sequencer preemptively, which can save a lot of resources. The pipeline proposed shows that it correctly manages to identify plants, while keeping a low execution time to result ratio. This report shows that real time analysis is possible, and leads to accurate results.**

I. INTRODUCTION

The identification of a plant can be performed by analysing its DNA. Multiple extraction methods can be used to extract the DNA, to sequence it, and to extract the information from the sequencer. This report focuses on accelerating the result of the latter process. Before the sequencer has finished its job, analysis can start and extract the data to interpret it. This is useful not only because the results of the analysis are available sooner, but also because in the case that the analysis is satisfactory, the sequencer can be turned off preemptively. For the minION sequencer used in the context of this report, stopping it sooner has the benefit of saving a lot of resources, as each flowcell has a limited lifetime. Those flowcells can be expensive, meaning that saving their use has a big economical impact. In this report two pipelines will be introduced in order to try to get results as soon as possible out of the sequencer, while obtaining the most accurate identification of the plant.

A. Background research

This paper builds on the idea introduced by Awen Kidel Pena–Albert and Emilien Ordonneau in their paper "Optimizing Computational Processes and Offline Operations in a Bio-Informatics Nanopore Sequencing Pipeline"[1]. In their paper, they introduced a pipeline that could detect the plant being sequenced in real time. They also proposed a faster way of obtaining a consensus from a large number of reads, called the 80-20 method.

B. Contribution

In the pipeline proposed by Awen P. and Emilien O., the iterations where isolated from each other. The first contribution of this report will be to propose a pipeline where the information of one iteration can contribute to the next, enabling an analysis that isn't only based on one single iteration. Also, their pipeline only performed detection, and not identification of plants. In this report, the quality of the consensus using depth coverage will be included in order to enable identification of plants. The last improvement over their work is that a constant time pipeline will be proposed, to counter the problem of the pipeline lagging behind the sequencer.

II. METHOD

A. GENOROBOTICS WORKFLOW

It is important to node the context where the pipelines will be used in, as it influenced how they were designed. The goal is to identify a series of plants. In our context, it was decided that on average 12 plants will be sequenced per sequencing run. In order to identify them, four amplicons of their DNA will be amplified, namely matK, rbcL, ITS and psbA-trnH. The four amplicons of each plant are then assigned a barcode using the rapid barcoding kit 24. All the amplicons of all plants are then placed on a minION FLONGLE using a MinION Mk1B sequencer. When they come out of the sequencer, they are first demultiplexed using the barcodes, and then demultiplexed again using a custom script to form one batch of reads for each amplicon for each plant. The pipeline then takes one batch of reads and tries to detect from what species the amplicon comes from. Using the result of the 4 amplicons will then lead to the final result for the identification of the specific plant. This means that there will be 48 pipelines running per sequencing run.

The sequencer used is MinION Mk1B. MinKNOW is then used to interface with it. As parameters we ask for the fastq reads to be outputted every minute. Based on the values explained above and the setup, and on representative sequencing runs in the past, it was determined that on average each of the 48 pipelines would receive 18 reads every minute. Such waves of 18 reads will be called iterations henceforth. It is important to note that the setup used means that there are very few reads per iteration. The worflow can be modified to increase the number of reads, such as using a different flowcell or reducing the number of plants.

B. POST-RUN ANALYSIS

In the standard context of post-run analysis, there are four main stages performed, as shown in figure 1. The first step is to pre-process the reads. This is necessary as the raw reads coming from the sequencer can sometimes contain defects that are introduced by the sequencing technique. For instance a DNA strand could form a loop with itself, and hence the sequencing pore would read the DNA endlessly without stopping. Also, sometimes some of the reads don't contain enough information to be included in the later steps of the analysis. Those would simply add more noise and more computation, hence make the analysis harder. More precisely, first the reads shorter than 300 bases and longer than the size of their amplicon are discarded. Then, reads with an average quality of 10 or less are discarded. This is based on Oxford Nanopore's recommendation. Then, reads with a base with a quality higher than 60 were removed, as those were identified to be defects due to the sequencer. Lastly, the reads are pairwise aligned against a reference read (the longest one found so far) to make sure that they are all on the same DNA strand (not complementary to each other) and in the same direction (not in reverse).

Since the reads outputted from the sequencer always contain some basecalling errors, a consensus is then created from those reads to reconstruct the original amplicon being sequenced. For that end, we first create a draft consensus using SPOA[3] to generate a multiple sequence alignment. Then, that draft is fed into Medaka[2] along with all the original reads to improve the draft using the read quality scores. This combination of tools was chosen as it gave the best results.

The third step is to check the quality of the consensus created from the pre-processed reads. The metric most often used in the scientific community is to use the depth coverage of each base along the consensus. The standard threshold used is that each base in the consensus must have a coverage of minimum x40. Once that threshold is reached, the consensus is deemed of high quality. The metric is alculated using Mosdepth[4]. While a higher coverage will mean more confidence in the bases of the consensus, in this analysis obtaining the threshold will be deemed sufficient.

In the case where we don't know the reference sequence of the consensus, we can't only use depth coverage for the quality of the consensus. For example, we could sequence an amplicon of 700 bases, but receive reads from the sequencer that create a consensus of 400 reads. In that case, reaching a coverage depth of x40 will mislead the confidence we have in the consensus. This is why in this paper a new metric is used, namely multiplying the median coverage of the consensus with the length of the consensus. More precisely, the length of the consensus doesn't include the start and end of the sequence that has a coverage below the median coverage of the consensus. This is done as we don't yet have enough confidence in that region to include it in the consensus. With this new metric for the consensus quality, if the median coverage increases or if the length of the consensus increases, then we know we can have more confidence in the consensus generated.

The fourth step is to use the consensus to try to detect what species it belongs to. For that, a local blastn query is executed against a NBCI database of that amplicon. The match with the best score is then taken, and its query coverage and percentage identity is stored to better interpret the result.

It is important to note that measuring the quality of the consensus is essential in our context. Potentially unknown plants (or previously never sequenced and uploaded to NCBI) will be used with the pipeline, and it should be able to detect those cases. Thanks to the quality of the consensus, we can better interpret the result of the detection. If the quality of the consensus is low, we can't interpret anything from the detection. However if the quality of the consensus is high, and the detection doesn't give any match, it could be the case that the plant is new.

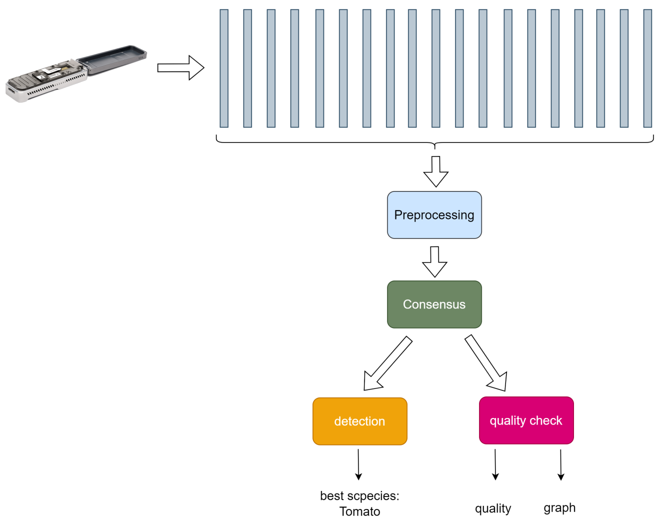

Fig. 1. This diagram shows the main stages in a post-run analysis context.

C. *REAL-TIME ANALYSIS*

During real time analysis, we will interpret the data outputted from the sequencer as it is running. Hopefully, we will be able to reach the desired conclusions about the data quickly and hence stop the sequencing sooner, saving some resources. The main goals that the pipelines should have are: 1. constant time of computation for all iterations. This is because we want to make sure that even if we have to let the sequencer run for a long time, that the pipeline can keep up with it. If that isn't the case, then we would at some point reach the moment where the sequencer keeps outputting data while the pipeline lags behind, giving outdated results. 2. Get a result outputted as soon as possible, and as frequently as possible (as soon as the first iteration starts and outputted at every other one). 3. Have a cost to result ratio that is the lowest. The goal is to spend as little resources as possible to get the best result as possible. This will influence how many pipelines can run at the same time, which we want as many as possible.

D. *NAIVE PIPELINE*

The naive way of obtaining this real time results is to perform a post-run analysis after each iteration on all the reads received so far, as shown in figure 2. The main issue with this approach is that the iterations don't have a constant time of computation, due to the ever increasing number of reads given as input to the post-run analysis. At some point, the pipeline will be slower than the sequencer, so the pipeline will be ignoring new reads and hence it will give sub optimal results compared to what the sequencer actually outputs. Although the 20-80 algorithm of Awen P. and Emilien O.

can heavily alleviate the execution time of the consensus, it still remains a polynomial time algorithm in terms of the size of the input, and as the number of reads increase at each iteration, the pipeline is bound to be caught up by the sequencer at some point.

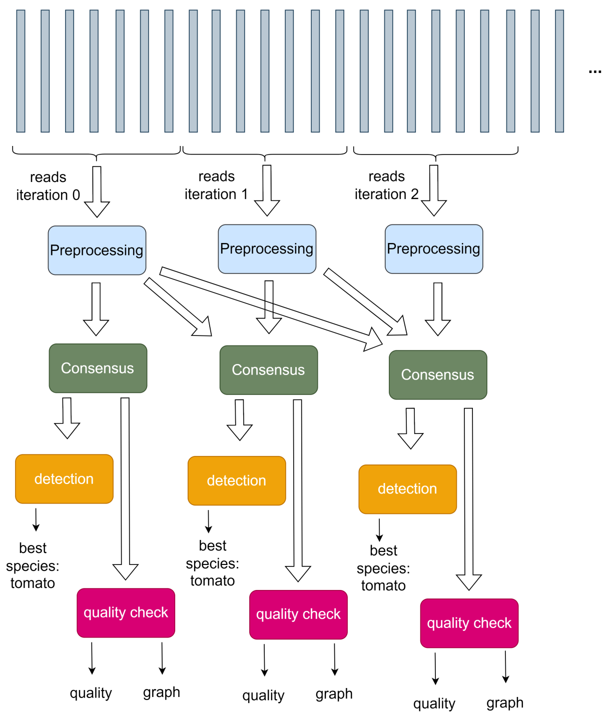

Fig. 2. This diagram shows the main stages of the naive pipeline.

However, this process should give the best results for the creation of the consensus, as all reads are taken into account. This way, all the information from the sequencer is used for creating the consensus, and hence this should mean that the consensus better represents the reads.

E. BEST-X PIPELINE

This pipeline was inspired by the fact that reads that come from the sequencer have a great variety of qualities. Some of the reads that come out of the sequencer have very few basecalling mistakes and have almost the exact length of the amplicon. Having this in mind, one strategy of recovering the amplicon of the plant being sequenced is to create a consensus on those best quality reads only. Since they are very close to the amplicon being sequenced, with little work they should give great results.

The pipeline works by keeping a running sample of the best reads received so far from the sequencer, and will only work on those to create the consensus and detection. If the reads in the sample are of very good quality, then we can keep only a few of them, making the whole work of the consensus and quality check much lighter, while obtaining good results for the consensus and hence also for the the detection. Here "quality" of a read means 2 features: their average basecalling quality score, and their length. Those two factors are multiplied with each other to get a score for each read. Using both factors was chosen as it gave the best results.

The steps of the pipeline are shown in figure 3. As we can see, an extra step is added where only the best x reads are chosen between the new reads of this iteration coming from the sequencer and the best x reads of the previous one. This makes the sample go from one iteration to the next, while being improved if better reads come from the sequencer. Performing this step is takes constant time as there is a fixed number of new reads coming from the sequencer (18 in our case) and at most x from the previous iteration. Then, a post-run analysis is performed on the reads in the sample only.

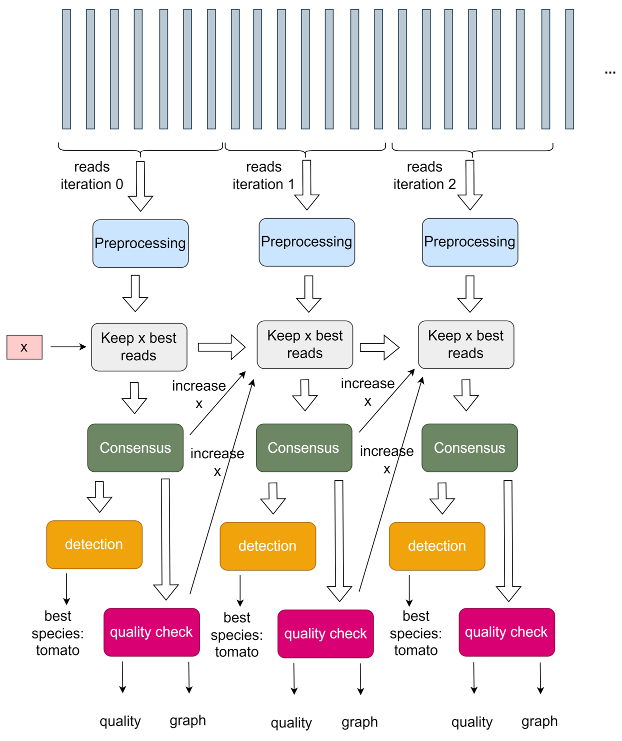

Fig. 3. This diagram shows the main stages of the best-x pipeline. One parameter should be supplied, namely the initial value of x.

There is one parameter that must be fed into the pipeline. It is the initial value of x. In essence, if x is very large, then the pipeline would behave just like the naive pipeline above, since all the new reads would always be added to the sample. Ideally the value of x should be as small as possible, this would quickly limit the increase of new reads in the sample, and hence keep the analysis constant and light. However, the initial value of x is lower bounded by the minimum depth coverage. The value of x can't be lower than 40 in this context. Also, setting x close to that limit will mean that the pipeline will struggle to reach a consensus with a depth coverage of exactly x40 along its whole length (if the reads are small compared to the reference amplicon). Hence, in practice it was determined that 1.5 times the minimum depth coverage as initial value of x gave the best results. Hence, in this report x will be initialised to 60.

The advantage of this pipeline is that a value of x that is set too high will not negatively influence the quality of the results. The only downside is that the pipeline will take more time per iteration to be executed. Hence, the value should be set initially to the lowest value (as explained above), and increased if necessary. The value of x could be too low if the reads of the sequencer are of very bad quality. The first point where we can notice that is if medaka doesn't manage to create a consensus. It can complain and return an error if the reads don't yield a single consensus. The second point where we can notice that x is too low is if the median depth coverage doesn't increase over time anymore, and is stuck below the desired minimum depth coverage. These two criteria can be used as feedback to tell the pipeline to increase the value of x in the next iterations.

Thanks to this choice of initial value of x being as small as possible, and these two feedback loops, the pipeline can stay lightweight at the start when the reads have a very high quality, and can adapt automatically when it detects that the reads have low quality. This implies that the pipeline will only perform a heavy analysis if absolutely necessary, improving the time of execution to the result ratio that is desired.

Another advantage with this pipeline is that it can automatically detect when it is satisfactory to stop the sequencing. As explained above, the metric used for the quality of the consensus combines the median depth coverage and the length of the consensus. The median depth coverage is expected to increase and stabilise around x. However, the length of the consensus is unknown, and will stabilise around the actual length of the amplicon being sequenced. When the metric of the quality of the consensus reaches a maximum point, then this shows that the consensus has found its desired length. The early stopping criterion is achieved when the metric of the quality of the consensus stabilises (increases by less than 5% compared to 8 iterations behind), and when the median depth coverage is above the minimum depth coverage. When those two conditions are met, the pipeline stops as we know that we have achieved a consensus with the desired minimum depth coverage and that has achieved the correct length.

III. Results

In order to test the pipelines, three datasets were used. The first was for the plant *Allium Ursinum* for the amplicon ITS, the second was *Ficus religiosa* for the amplicon psbA-trnH and lastly *Solanum Lycopersicum* for the amplicon rbcL. Those datasets were used as they span the three different genes, because they were sequenced from a real expedition carried in a botanical garden which made their data more realistic, and lastly because the quality of those sequencing runs were quite good. Those datasets will be considered as good quality reads for comparison purposes.

In order to simulate worse sequencing runs, those three datasets were then downgraded. There are three ways the sequencer can give worse output. The first is when the quality of the reads is lower. This was simulated by decreasing the average read quality to be closer to 10 (the minimum allowed according to Oxford Nanopore). The second way the sequencer can give worse output is when the DNA strands were cut in half more often, and hence the reads are shorter. To simulate that a portion of the reads were cut in half (and their second half discarded to make sure there is the same amount of reads as before). The third difference the sequencer can make is to output less reads than usual. This can often happen, especially during the end of a lifetime of a flowcell, when the majority of the pores have become inactive. Such modification wasn't simulated in the "worse" datasets, since that would simply result in the pipeline taking more time to find its result, which can be easily calculated by comparing it to the results below. The three datasets were then modified using the first 2 factors, until EPI2ME couldn't find a consensus anymore, confirming that those reads were downgraded to the maximum. These 3 new datasets would then be useful to check how the pipeline would react with very bad quality reads. The distribution of their read quality and read length can be found in the appendix.

A. *Quality analysis*

A first interesting test for the best-x pipeline is to analyse how the average read quality in the sample evolves over time. The metric used for quality here is the one used for choosing what the "best" reads are to keep in the sample, namely their length and their average basecalling quality. As we can see in figure 4, the reads are initially of bad quality, but they quickly shoot up to reach higher values. It seems that it then levels out and doesn't increase a lot anymore at some point. As we can see, this pattern is visible for the three good quality datasets, but also the bad quality ones. This implies that after some point the reads in the sample don't change a lot, and that the amount of information that the sample contains doesn't increase a lot anymore. This could indicate that the pipeline should stop.

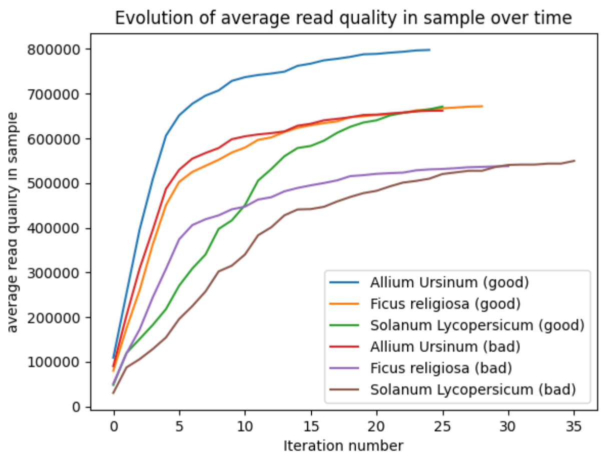

Fig. 4. This graph shows the evolution of the average quality of the reads in the sample over time for the 6 datasets.

Then, the quality of the consensus can be plotted over time. The quality here is the metric explained above. As we can see in figure 5, the consensus quality also quickly increases at the start and then it slows down at the end. This pattern is visible for all datasets, even for the bad quality ones. This shows that the pipeline manages to reach a stable point where the median of the depth coverage and the length of the consensus is stable. As we can see, the pipeline correctly managed to detect when the equilibrium was reached and managed to stop.

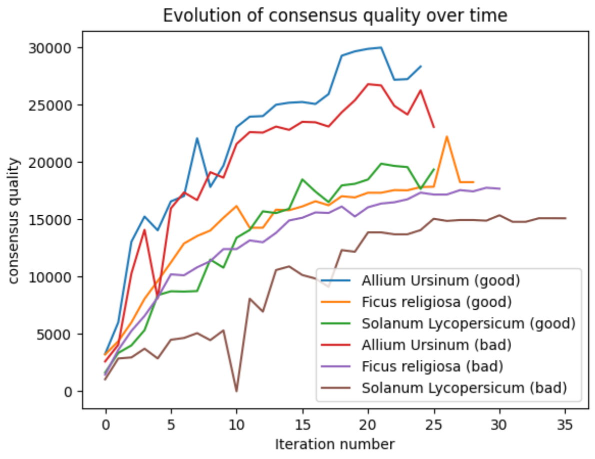

Fig. 5. This graph shows the evolution of the quality of the consensus over time for the 6 datasets.

What is important to note is that the quality of the consensus is very correlated to the quality of the reads in the sample. Figure 6 illustrates the average quality of the reads and the quality of the consensus over time for the dataset "*Allium Ursinum* (good)". As we can see, the correlation is very strong between the two metrics. This indicates that the pipeline correctly manages to improve the quality of the consensus as the reads in the sample improve in quality over time. This shows that the pipeline doesn't stall at some point, and correctly manages to improve its results. Also, it shows that when the quality of the consensus slows down, it means that the quality of the reads slowed down, which shows that the pipeline correctly manages to detect when it is not advantageous anymore to continue sequencing.

As explained above, the pipeline manages to reach a stable point in terms of the quality of the consensus. Figure 7 shows the evolution of the depth coverage for the dataset "*Allium Ursinum* (good)". As we can see, the depth coverage is initially too low, but it quickly shoots up to reach the desired minimum depth coverage of x40. The depth coverage of the last iteration shows that it managed to reach this threshold for almost the entirety of the consensus, showing that the pipeline managed to get a very good consensus quality when it stopped. This behaviour was also observed for the rest of the datasets. What is interesting is that the difference between the depth coverage of the 3rd and 10th iteration is much bigger than the difference between the 10th and 24th. This is confirmed on the graph of the evolution of the consensus quality, where it starts to level off after iteration 10.

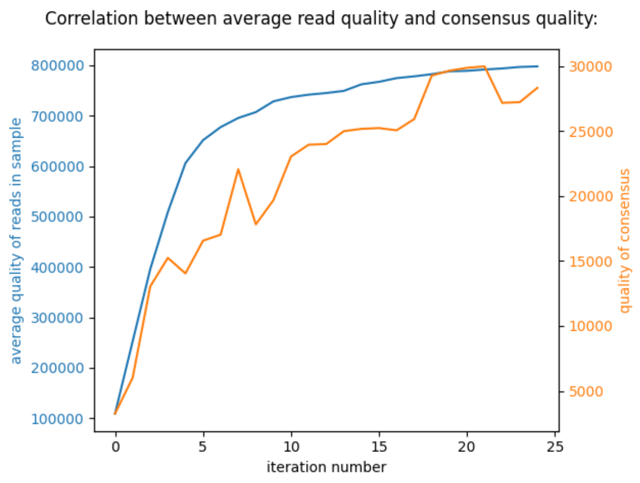

Fig. 6. This graph shows the correlation between the average quality of the reads in the sample and the quality of the consensus over time.

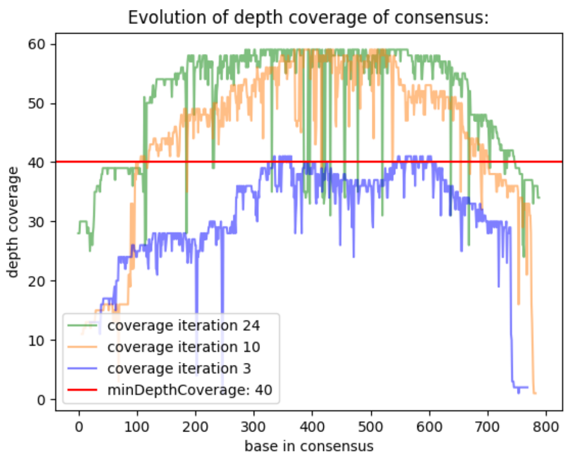

Fig. 7. This graph shows the evolution of the depth coverage of the consensus created for three iterations.

One thing that must be done is to check that the consensus of the sample of the best-x pipeline represents the other reads well. To do that, the consensus that was created on the last iteration of the best-x pipeline was compared to a consensus of all the reads up to that point. As we can see in table I, the length of the consensus of the sample is always longer than the consensus of all the reads, showing that the consensus of the best-x sample is better in that regard. We can also see that the percentage identify of the two consensus is very high, perhaps except for the last dataset. Though we can see that the query coverage is not that good. Upon closer inspection, it was determined that the gap in the sequence was always to the right or left of the other sequence. This shows that the two consensus were simply a bit misaligned, one contained a bit more information in the start of the sequence while the other did the opposite. These results show that in general the consensus created from the sample in the best-x pipeline doesn't deviate too far from the true one, as desired.

Dataset	len. cons. best-x	len. cons. all	query cov.	% ident.
Allium Ursinum (good)	792	605	80%	95%
Ficus religiosa (good)	586	548	96%	92%
Solanum Lycopersicum (good)	643	604	90%	97%
Allium Ursinum (bad)	785	719	81%	96%
Ficus religiosa (bad)	564	540	82%	92%
Solanum Lycopersicum (bad)	666	530	74%	80%

TABLE I. This table shows the comparison of the consensus of the best-x pipeline with the consensus of all reads.

The correctness of the detection was also analysed. Table II shows the result of the detection for the last iteration of the best-x pipeline. As we can see, for all datasets the species that was detected was correct. The percentage identification of the hit is very high, perhaps with the exception of the 4th dataset. Though, the query coverage is not perfect. The area that didn't match was always at the start and end of the consensus. This was probably due to the low depth coverage of the consensus in those areas. In general this table shows that in the context of plant detection, the result is very accurate, but that there could be some improvements in terms of the query coverage.

Dataset	Species	query cover.	% identity
Allium Ursinum (good)	Allium Ursinum	82%	97%
Ficus religiosa (good)	Ficus religiosa	78%	98%
Solanum Lycopersicum (good)	Solanum lycopersicum	73%	97%
Allium Ursinum (bad)	Allium Ursinum	81 %	85%
Ficus religiosa (bad)	Ficus religiosa	75 %	92%
Solanum Lycopersicum (bad)	Solanum Lycopersicum	89 %	100%

TABLE II. This table shows the result of the detection for the last iteration of the best-x pipeline.

B. performance

As explained above, the execution time of the pipelines is crucial. The dataset "*Allium Ursinum* (good)" was used to measure the execution time of the naive pipeline for each iteration and each step within. As we can see in figure 8, the pipeline takes about 11 seconds for the first iteration, and about 11 for the last one. As we can see, the naive pipeline managed to keep a constant execution time. This is a surprising result since the algorithm used for that pipeline receives more reads each iteration, and is an algorithm that has a polynomial time execution in terms of the number of reads. What this indicates is that in this setup with 18 reads per iteration the increase in execution time remains very small. This indicates that infuse the naive pipeline gives good performance results when few reads come from the sequencer. Another test was then performed where 200 were outputted for each iteration. As we can see in figure 9, the execution time at the start is about 12 seconds, while at the end it is about 19 seconds. This confirms the non-constant execution time of the pipeline. What we can also see is that the majority of the time spent is to create the consensus, and that it is the one responsible for the increase in time.

When can now analyse the execution time of the best-x pipeline on the same dataset (and with again 18 reads per iteration). As we can see in figure 23, the pipeline takes about 10 seconds for the first iteration and about 10 for the last.

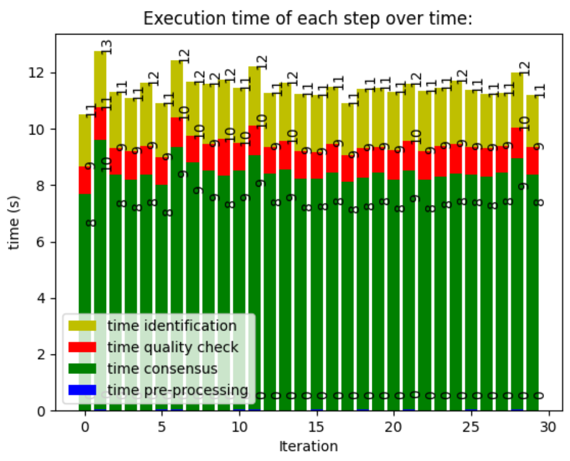

Fig. 8. This graph shows the evolution of the execution time for the naive pipeline, with 18 reads per iterations.

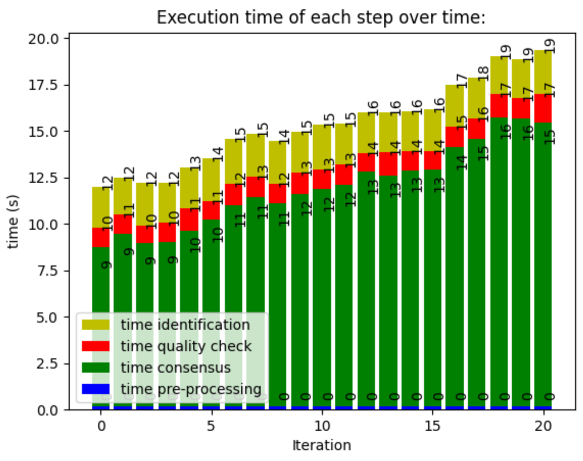

Fig. 9. This graph shows the evolution of the execution time for the naive pipeline, with 200 reads per iterations.

There is a bit of fluctuations in the results, but we can see that in general the execution time remains constant at around 11 seconds, as desired. We can see that the consensus is also the step that takes the majority of the time for each iteration. As before, we can confirm this constant execution time by increasing the number of reads per iteration from 18 to 200. As we can see in figure 11, the execution time started at about 10 seconds and remained at 10, as desired.

IV. Discussion

As we have shown above, the analysis on the best-x pipeline first showed that the quality of the reads in the sample increases over time. Then, it was shown that this increase in quality of the reads was correlated with an increase in the quality of the consensus. The early stopping criterion was then verified by confirming that when the pipeline stopped the depth coverage was very high along the consensus. It was also shown that the consensus generated from the sample is quite close to the one generated from all

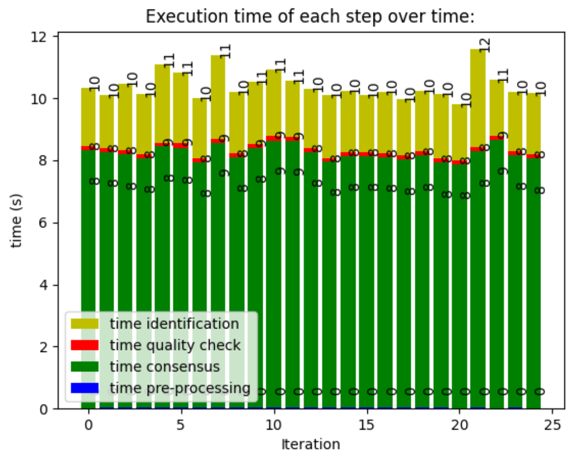

Fig. 10. This graph shows the evolution of the execution time for the best-x pipeline, with 18 reads per iterations.

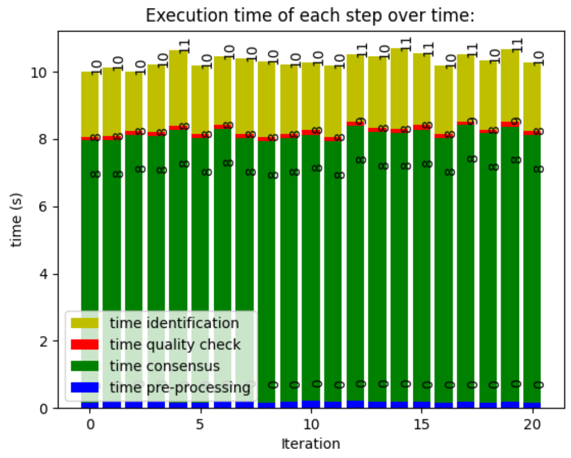

Fig. 11. This graph shows the evolution of the execution time for the best-x pipeline, with 200 reads per iterations.

the reads. Finally, the accuracy of the detection was shown to be very good, though the percentage coverage was not ideal. In terms of performance, it was demonstrated that the naive pipeline fared well with a low number of reads per iteration, and that the best-x pipeline managed to keep a constant execution time.

The best-x pipeline does have a drawback though, namely that it might stop later than necessary. This is because the depth coverage of the consensus is only computed on the reads of the sample (to keep a constant execution time). As time goes on, the depth coverage calculated will continue to underestimate the true depth coverage of all the reads. Hence, when the pipeline thinks that it has finally achieved the minimum depth coverage using the reads in the sample, perhaps it has actually achieved it sooner if it calculated the depth coverage of all the reads received from the sequencer. How much the pipeline lags behind this real true coverage wasn't measured, and remains something to be taken into account in order to further improve the efficiency of the resources available.

A factor that could have influenced the results in this report is the removal of reads shorter than 300 bases. While it is a step performed on EPI2ME, and hence should be a sensible choice, perhaps it heavily influenced the results in this report. Perhaps using a much lower minimum read length could still give good results without adding too much noise, and hence could provide results faster, again improving the efficiency of the resources.

While the results shown in terms of the detection here are promising, it is important to note that only 3 real datasets where tested (and 3 generated from the original ones). In order to have a better understanding of the overall performance of these pipelines, analysis should be performed on more datasets ranging across all the 4 amplicons to make sure that the results obtained here are representative of the rest.

Another point to mention is that the performance analysis showed that both pipelines took about 10 seconds per iteration (at the minimum). This means that in theory only 6 pipelines could be run at the same time before the next wave of reads are outputted by the sequencer. While this does seem quite low compared to the desired amount of 48, there are a lot of optimisations that could bring this value higher. For instance the pipeline could run as separate processes on the CPU and hence run in parallel up to the number of cores on the computer running the sequencing. Also, the code was written in python, which add a big overhead. Optimising the code should be able to bring it down to more manageable values.

V. Conclusion

In this report, a pipeline was proposed that enables fast identification of plants while keeping a very low cost of execution to result ration. It was shown that the results of the quality of the consensus manage to reach the desired minimum depth coverage and that the results of the detection are quite good. There remains one thing to be improved though, which is the query coverage of the detection. The naive pipeline was also shown to give good performance results in cases where there were few reads from the sequencer.

References

[1] Awen Kidel Pena–Albert and Emilien Ordonneau, “Optimizing Computational Processes and Offline Operations in a Bio-Informatics Nanopore Sequencing Pipeline”. Genorobotics 2023.
[2] https://github.com/nanoporetech/medaka
[3] https://github.com/rvaser/spoa
[4] https://github.com/brentp/mosdepth

APPENDIX

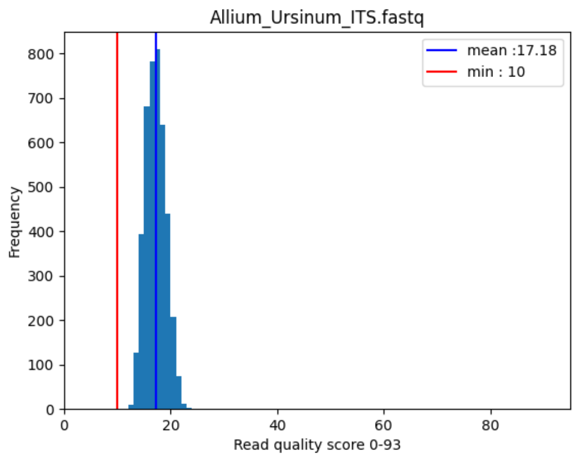

Fig. 12. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Allium Ursinum* (good)".

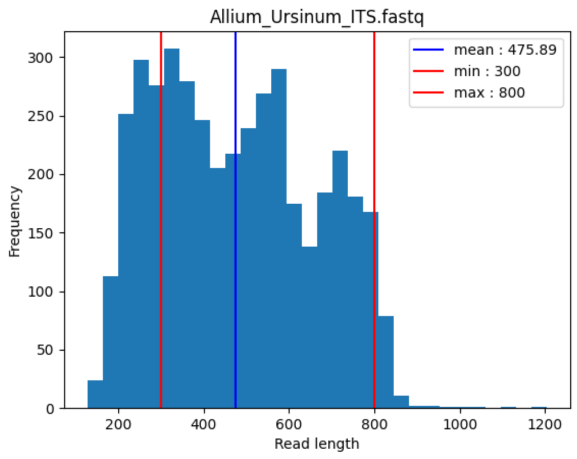

Fig. 13. This graph shows the distribution of the length of the reads in the dataset "*Allium Ursinum* (good)".

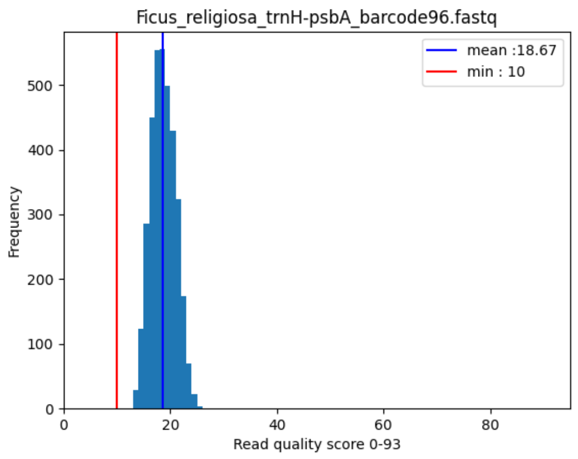

Fig. 14. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Ficus Religiosa* (good)".

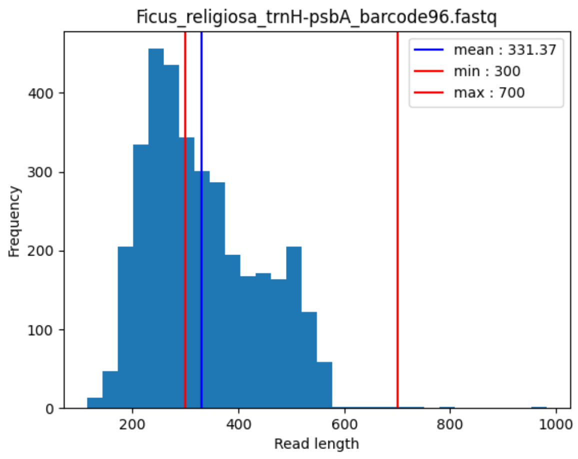

Fig. 15. This graph shows the distribution of the length of the reads in the dataset "*Ficus Religiosa* (good)".

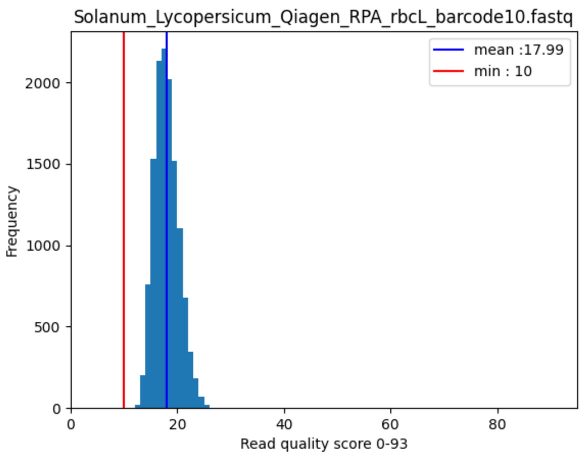

Fig. 16. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Solanum Lycopersicum* (good)".

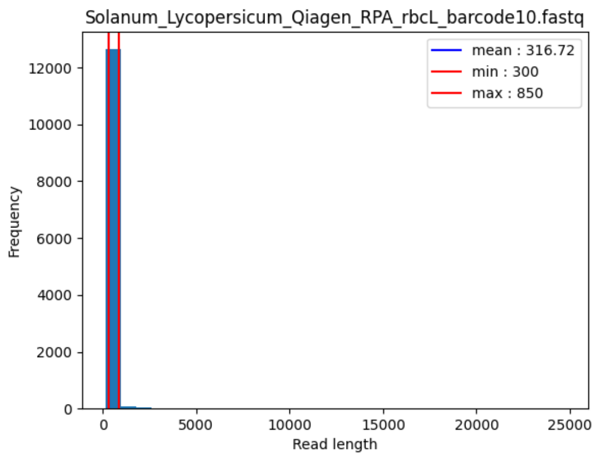

Fig. 17. This graph shows the distribution of the length of the reads in the dataset "*Solanum Lycopersicum* (good)".

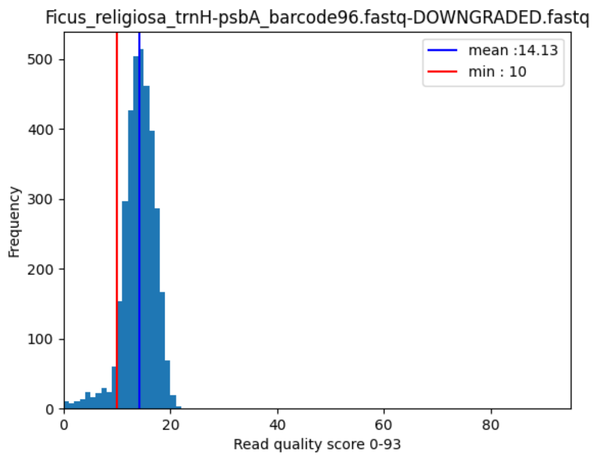

Fig. 20. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Ficus Religiosa* (bad)".

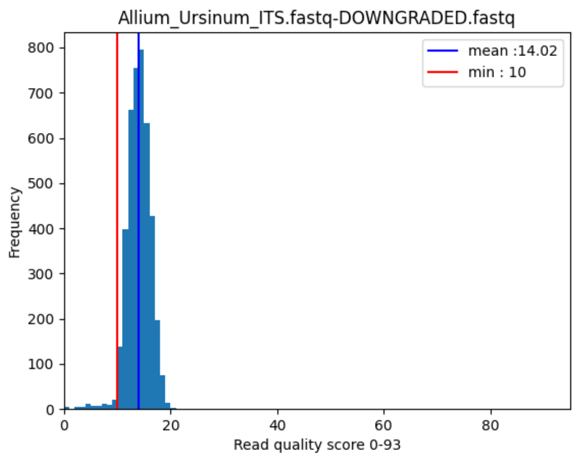

Fig. 18. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Allium Ursinum* (bad)".

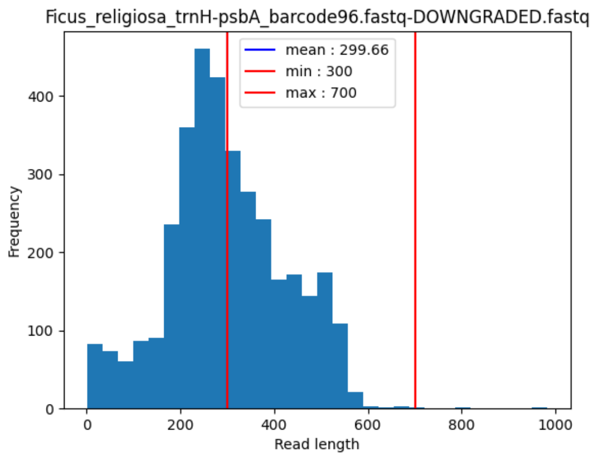

Fig. 21. This graph shows the distribution of the length of the reads in the dataset "*Ficus Religiosa* (bad)".

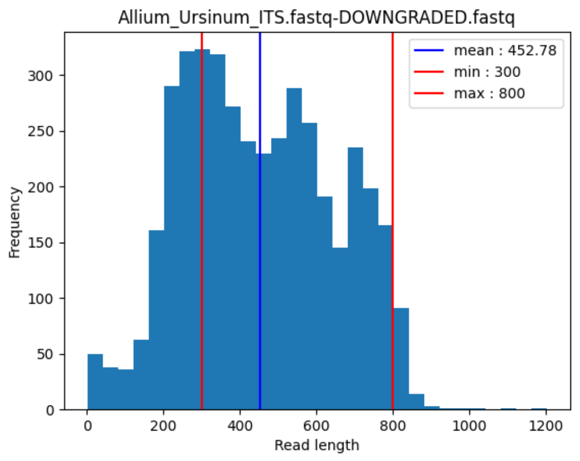

Fig. 19. This graph shows the distribution of the length of the reads in the dataset "*Allium Ursinum* (bad)".

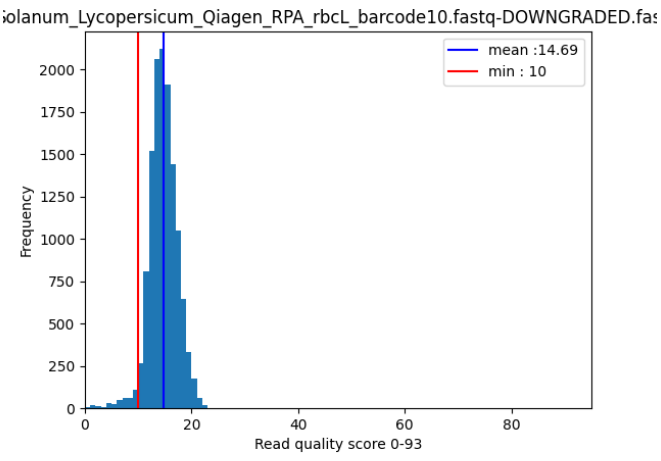

Fig. 22. This graph shows the distribution of the basecalling quality of the reads in the dataset "*Solanum Lycopersicum* (bad)".

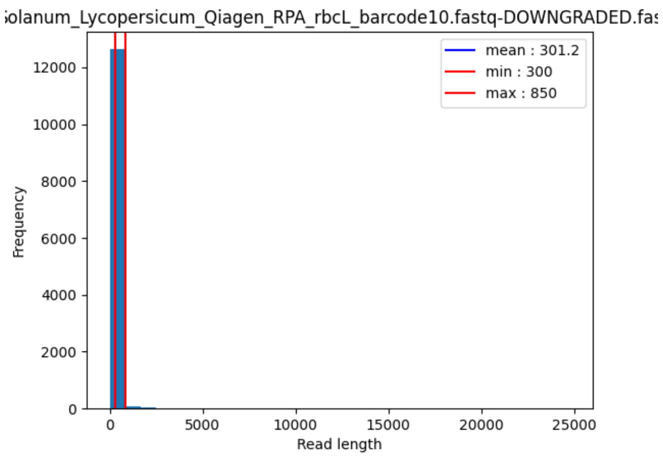

Fig. 23. This graph shows the distribution of the length of the reads in the dataset "*Solanum Lycopersicum* (bad)".