UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Metagenomic classification of long reads with overlap graphs

Prof. Matteo Comin, Eng. Mattia Luciani, Eng. Margherita Cavattoni

Metagenomics Binning · LongReads · Read-Overlap Graph.

Abstract

Motivation: Current technologies allow the sequencing of microbial communities directly from the environment without prior culturing. The major problem when analyzing a metagenomic sample is to taxonomically annotate its reads to identify the species they contain. Most of the methods currently available focus on the classification of reads using a set of reference genomes and their k-mers. While in terms of precision these methods have reached percentages of correctness close to perfection, in terms of recall (the actual number of classified reads) the performances fall at around 50%. One of the reasons is the fact that the sequences in a sample can be very different from the corresponding reference genome, e.g. viral genomes are highly mutated.

Methods: To address this problem, in this paper we propose ClassGraph 2.0, a metagenomic taxonomy refinement tool that makes use of reads overlap information from the reads overlap graph, to refine the results of existing tools to classify unlabelled reads. ClassGraph 2.0 needs two types of input: one is the reads overlap graph and the other is the output of a binning tool. At this stage the graph is stored in a data structure, where to each vertex/read is associated a label given by the binning tool. The arcs in the graph are weighted based on the overlap of the two reads. Under the assumption that connected reads are more likely to be from the same species, we refine the vertex labels in the reads overlap graph. This procedure is performed to search for nodes that are mislabelled by the binning tool. To do this, a RefineLabel algorithm is used, which can eliminate incorrectly assigned labels, followed by a LabelPropagation algorithm that expands the correct labels. The RefineLabel algorithm counts neighboring nodes with the same label as the node under examination. If the label of the node is different from most of the labels of neighboring nodes, then this label must be removed. Once all nodes have been processed, the labels are deleted. In the label propagation phase each labeled node sends its label to its neighbor, along with the weight of the arc connecting the two nodes. The receiving node will choose its label maximizing the score of the associated arcs.

Results: We tested ClassGraph 2.0 on three simulated datasets of long reads, created using SimLoRD with 8, 20 and 50 species as in [cite], and a real marine metagenome with 5000 species, from the CAMI2 challenge, for which the ground truth is known. We chose Kraken2, which is the state of the art, for the taxonomic classification of reads. We compared the classification performance of Kraken2 with ClassGraph and ClassGraph 2.0. Sensitivity, precision, F1-Score and PCC were used to assess the accuracy of the classifications. Instead, time and memory were used to assess the running costs of the tools.

	Sim - 8	Sim - 20	Sim - 50	Marine
Kraken2	Sens: 0.765628 Prec: 0.861202 **F1: 0.810608** PCC: 0.996426 Time: 00:11:49 Memory: 47.45 GB	Sens: 0.629978 Prec: 0.776203 **F1: 0.695488** PCC: 0.923726 Time: 00:12:37 Memory: 47.47 GB	Sens: 0.570512 Prec: 0.704239 **F1: 0.630361** PCC: 0.946056 Time: 00:28:06 Memory: 47.53 GB	Sens: 0.577274 Prec: 0.806809 **F1: 0.673009** PCC: 0.989780 Time: 00:12:31 Memory: 47.43 GB
ClassGraph	Sens: 0.789763 Prec: 0.864731 **F1: 0.825549** PCC: 0.995107 Time: 00:01:07 Memory: 2.25 GB	Sens: 0.666136 Prec: 0.780303 **F1: 0.718714** PCC: 0.933724 Time: 00:04:26 Memory: 11.15 GB	Sens: 0.596122 Prec: 0.682097 **F1: 0.636218** PCC: 0.952594 Time: 00:10:58 Memory: 28.56 GB	Sens: 0.697363 Prec: 0.814654 **F1: 0.751459** PCC: 0.989609 Time: 00:14:00 Memory: 26.18 GB
ClassGraph 2.0	Sens: 0.993279 Prec: 0.994949 **F1: 0.994113** PCC: 0.999963 Time: 00:01:28 Memory: 2.72 GB	Sens: 0.869446 Prec: 0.961895 **F1: 0.913337** PCC: 0.977861 Time: 00:05:46 Memory: 13.22 GB	Sens: 0.769968 Prec: 0.827907 **F1: 0.797887** PCC: 0.983608 Time: 00:13:36 Memory: 33.91 GB	Sens: 0.795554 Prec: 0.909551 **F1: 0.848742** PCC: 0.989793 Time: 00:17:27 Memory: 29.01 GB

Figure 1: Comparison of the results of Kraken, ClassGraph and ClassGraph 2.0 in the different datasets.

From the results in the Figure 1 it can be seen that after running ClassGraph the classification accuracy, in terms of F-measure, for all datasets increases slightly, mostly due to a better sensitivity. With ClassGraph 2.0 there is a further increase in the classification accuracy for all datasets, with a substantial increase of both sensitivity and precision. From the results it can be observed that, although Kraken2 is one of the best binning tools, it cannot classify all reads, in fact the sensitivity on the most complex datasets is 57%, and the precision ranges in [70%-80%]. With ClassGraph, and its Label Propagation algorithm, these labels can be expanded, increasing the number of classified reads, while preserving a similar precision. In ClassGraph 2.0, with the new Refine Label algorithm before the Label Propagation, the classification accuracy further increases. The performance of ClassGraph 2.0 shows that some of the classifications produced by Kraken2 are incorrect, but it is possible to recognise them and then expand only the correct ones, thus improving both sensitivity and precision. It can also be seen that the execution times of Kraken2 and ClassGraph 2.0 are of the same order of magnitude. The memory required to run ClassGraph 2.0 is less than that required to run Kraken2.

ClassGraph 2.0

ClassGraph 2.0 is a metagenomic taxonomic binning tool that uses the long read connectivity data from the long read graph to eliminate classifications made by the binning tool that may be incorrect and thus classify unclassified reads. In this way ClassGraph 2.0 can further improve classification accuracy. For the development of ClassGraph 2.0 we started from an existing tool working on short reads. In figure 2, we can see the differences between the workflow of ClassGraph and ClassGraph 2.0. Each of the steps illustrated will be described below.

ClassGraph workflow

ClassGraph (1) is a metagenomic taxonomic binning tool that uses reads connectivity data from the reads overlap graph to classify unlabelled reads. ClassGraph uses the binning result of an existing taxonomic classification tool and a LabelPropagation algorithm to predict the labels of reads that could not be classified. The new paradigm used by ClassGraph can improve classification accuracy even further.

Pre-Processing

ClassGraph requires two input files: one representing a graph of reads and the other containing the result of the classification process. The labelles assigned to the reads by the pre-existing binning tool will be propagated over the graph to the still unclassified reads. ClassGraph is thought to be used with both paired-end and single-end reads. ClassGraph has designed for use with paired reads. In this case the binning tools collapse each pair, producing as output a single reads with its label. SGA, on the other hand, keeps the reads separate, matching every read with a node in the graph. As two paired reads belong to the same DNA fragment, they are guaranteed to belong to the same species. For this reason in ClassGraph it was chosen to modify the SGA graph by collapse the nodes that represent paired reads. In this second step, the graph previously defined is simplified, reducing the number of arcs to an absolute minimum and the memory used accordingly. All arcs joining two nodes that have already been labelled are removed. This is made by the fact that they will never be used by the label propagation algorithm, which does not provide that labels can be modified once assigned. The graph is then memorised in a data structure, organised in arrays, which maintains the following information for every vertex:

1. Label l of the vertex if it has it, 0 otherwise.

2. List of arcs. Each arch is stored in the form of a pair (idv , p) where idv is the sequential identifier of the vertex at the other end of the arch and p is the weight

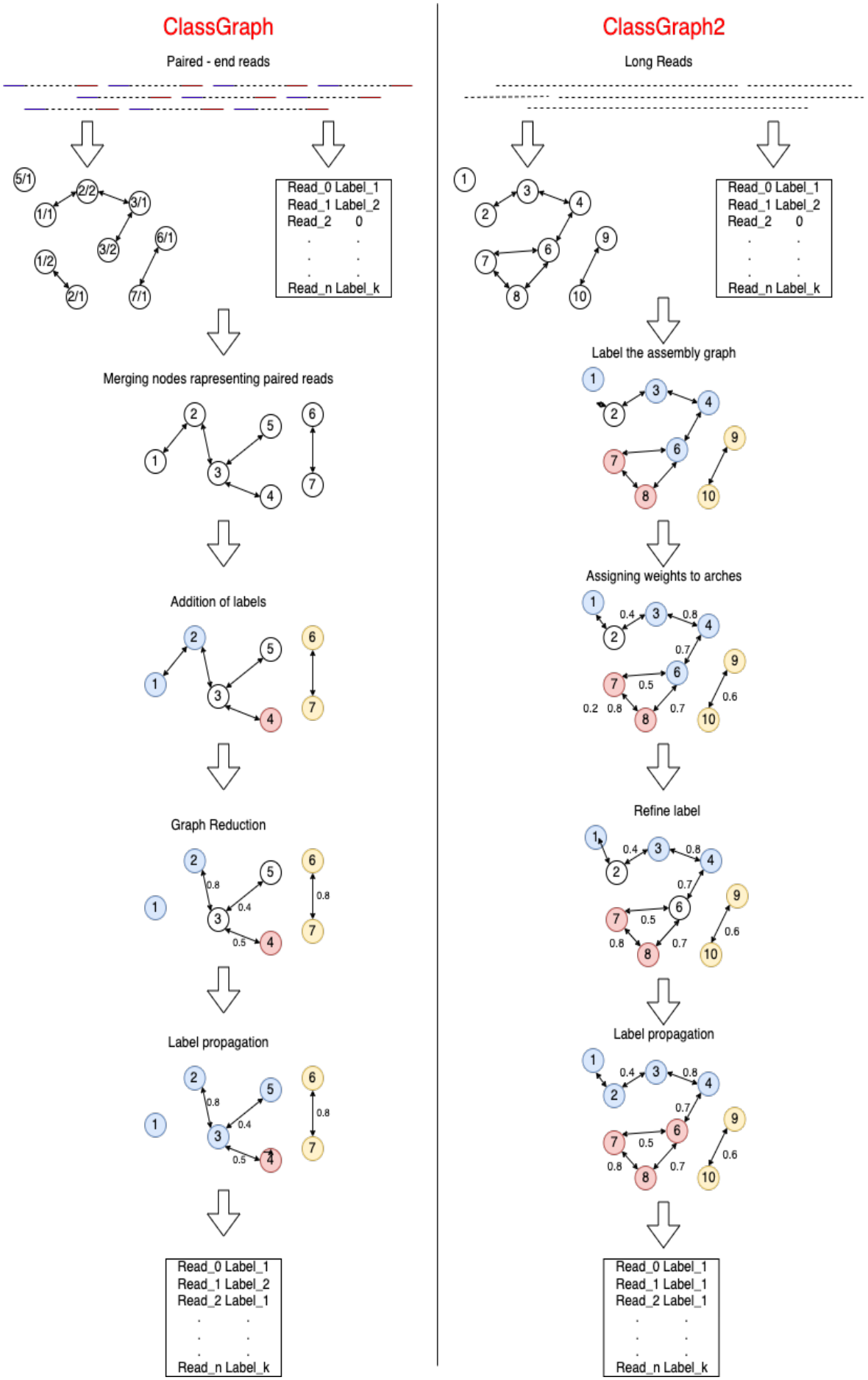


Figure 2: Comparison of the ClassGraph and ClassGraph 2.0 workflow

```
1: Level_0 = {All labelled nodes with at least one egde}
2: Level_1={∅}
3: for i = 1, 2, ..., #iter do
4:     for all j ∈ Level_{i-1} do
5:         for all edges e_jk do
6:             Send to the node k the label l_j with weight w_jk
7:             Add node k to Level_i
8:         end for
9:         Delete all the edges of node j
10:    end for
11:    for all j ∈ Level_i do
12:        For each label received l_k compute the sum of its weights
13:        Find the label l_max with the maximum weight
14:        l_j = l_max
15:    end for
16:    Level_{i-1} = Level_i
17:    Level_i=∅
18: end for
```

Figure 3: Label Propagation algorithm.

of the arch in question. All arcs connecting two vertices that already have a label will not be entered as explained.

Label propagation

Label propagation was implemented by using an iterative algorithm, the number of iterations of which was given as a parameter at the start of the tool. The decision to give a max_iteration threshold to this number was motivated by an empirical reason. In fact, it has been noticed that most propagations are carried out in the first few iterations. The idea behind the algorithm is that propagation proceeds by levels. In particular, at the $i-th$ iteration all and only those nodes that do not yet have a label and are connected by an arc directly to the nodes labelled at the $(i-1)-th$ iteration will be labelled. The minimum distance, in terms of number of arcs, between any labeled node at the $i-th$ iteration and its nearest node from those that had a label in the starting graph will then be equal to i. The more this distance is increased, the more error-prone the propagated labels will be. Hypothetically, by choosing a very large number, the propagation could possibly reach all nodes in the graph except isolated nodes or nodes belonging to an isolated group of nodes that do not all have a label. With the same max_iteration selected the two versions of LabelPropagation will label exactly the same nodes, but the labels assigned may differ.

Using the algorithm in figure 3, two for cycles are executed at each iteration to scan all vertices of the graph.

At the first scan for each node with a label $l \neq 0$, all its arcs are analysed. Each time an arc leads to an id_k node without a label, the pair (l, p_k) is added to the information of the id_k node, where p_k coincides with the weight of the arc in question. Once the arcs of an already labelled node have been scanned and possibly used, all its information, except for its label, is removed. This has the dual advantage of freeing up memory and not having to re-examine the same information on the next iteration. The second for loop decides which label to assign. To do this, it searches for which nodes are to be labelled, i.e. those which in the first for loop were added the information of type (l, p) transmitted by the nodes labelled in the previous iteration. When one of these nodes is found, we calculate, for each of the possible labels, which one maximises the sum of the weights of the arcs associated with it. This label will then be the one that will actually be assigned to the vertex in question. Each time the label is assigned, all the pairs (l, p) added to the node in the previous for are removed. It will therefore be the nodes labelled at this iteration that will propagate the labels to the next iteration.

ClassGraph 2.0 workflow

The idea behind ClassGraph 2.0, like ClassGraph, is to start from the results produced by a pre-existing binning tool (kraken2 (2)) and then try to extend them by assigning a TaxId also to sequences that had not been classified. As shown in figure 2, the pipeline starts with the data to be analysed as input to kraken2 and minimap2, from which the labelled reads and the graph are obtained respectively. The output thus obtained will then constitute the input for ClassGraph 2.0. In the case of ClassGraph, files in which the reads are presented in pairs are used to construct the graph. Each pair consists of two reads present at the ends of the same DNA segment. Here, compared to ClassGraph, we have the first major difference, namely the use of long reads in the input to construct the graph. Minimap2 (3) was used to construct it. Long-read sequencing, or third-generation sequencing, has several advantages compared to short-read sequencing. While short-read sequencing machines can produce reads of up to 600 bases, long-read sequencing technologies generate reads of more than 10 kb on a regular basis (4). Short-read sequencing is convenient, accurate and supported by a large range of analysis tools. However, natural nucleic acid polymers are eight orders of size in length, and sequencing them in short amplified fragments makes the task of reconstructing and counting the original molecules difficult. The advantages of long-reads technologies also make them ideal, for example, for accessing complex regions of the genome. This makes them ideal for clinical applications in diagnosis, prognosis and personalized medicine. Whereas ClassGraph works with the graph produced by SGA, an assembly program that falls into the category of de novo assemblers, ClassGraph 2.0 starts with the graph produced by minimap2 in .paf format, but for this tool you will not need all the

information contained in the outputs. In particular, to construct the graph, we need the vertices and the arcs connecting them. For the execution of LabelPropagation we also need the weights of the arcs, which in ClassGraph 2.0 corresponds to the normalisation of the chaining score given by minimap2. ClassGraph 2.0 follows, in some ways, the structure of pre-processing defined in ClassGraph but adds a RefineLabel algorithm that eliminates labels already assigned by the binning tool.

Label the assembly graph

At this point the graph is stored in a data structure organised in arrays, where for each vertex its label is stored. In particular, the vertex may have a label l, corresponding to the TaxId of the species, if the reading has been classified by the binning tool, or 0 if the binning tool has not classified the reads.

Assining weight to archs

While in ClassGraph the assignment of weights is done through the overlaps between reads divided by the length of the read, in ClassGraph 2.0 it is done differently. As already described minimap2 gives us a chaining score for each arc. This chaining score is divided by a constant in order to normalise the weights. After that the weights are added to a data structure which includes the nodes forming the arc and the weight.

RefineLabel

Based on the assumption that connected reads are more likely to be of the same species, we refine the vertex labels in the assembly graph. It is very likely that connected vertices or vertices located close to each other will belong to the same cluster since they are less distant (i.e. have more similarity) from each other. Thus, such vertices are expected to have the same label. The refinement process is done on the basis of the labels of such vertices. This procedure is carried out to find nodes that are incorrectly labeled by the binning tool.

The first implementation of RefineLabel searches for the neighbors of each node, after which it checks whether the label of the node under consideration is the same as that of its neighbors. In case they are different, both labels are removed, thus equal to 0. If one of the node labels under examination and that of the neighbors is equal to 0, the labels remain unchanged. In the figure 4, first image, you can see a hint of how it works.

The algorithm starts the search for the labels to be deleted from node number 1 and continues with the subsequent nodes up to 16. Whenever it finds a discrepancy between the label of the node and one of the neighboring nodes, it deletes both of them. From the graph, it can be seen that the node 1 has node 6 as a neighbor which has a different

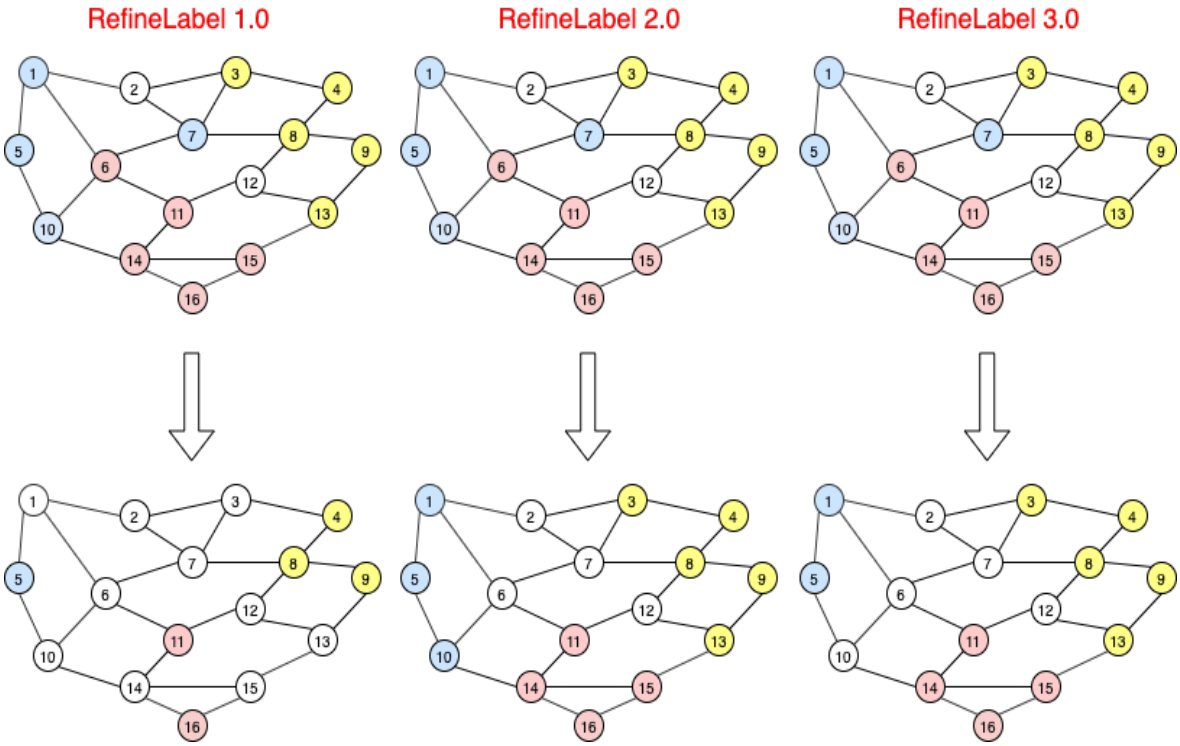


Figure 4: RefineLabel algorithms examples.

label, so both labels are deleted. Continuing with the iterations, nodes without labels are skipped. At node 3, we have a situation similar to that of node 1, so the elimination of the labels is carried out. Arriving at node 5 we notice that the neighbors are all without labels, so the node keeps its label. The algorithm continues as described to the last node by removing the labels.

The second version of RefineLabel was developed to try to reduce the number of erased labels. As the number of deleted labels is very high, it is possible that correct labels are also deleted. This version works very differently from the previous version. The algorithm counts neighbouring nodes with the same label as the node in question. Specifically, the algorithm for each node counts the labels of neighbours equal to that of the node under consideration. If the label of the node under consideration is different from the majority of the labels of the neighbouring nodes, it is immediately eliminated. If the number of neighbours' labels equal to that of the considered node and the number of neighbours' labels different from that of the considered node are equal, the considered node keeps its original label. As in the previous algorithm, nodes with no label are not considered. In the following figure 4, second image, we can see an example of its operation.

As can be seen, the algorithm leaves all node labels unchanged until it reaches node 6. Node 6 has as neighbors three nodes with different labels and one with the same label, so the label is eliminated. Node 7 has as neighbors two nodes with a label different from its own and two unlabeled nodes, so the label is dropped. The rest of the labels

are kept as original.

The third version of the RefineLabel algorithm is very similar to the previous version, but has one fundamental change. Whereas the previous algorithm, when it finds a label that may be wrong, deletes it immediately, this latest version makes a record of the labels to be deleted and then deletes them all together. With this solution, it is possible to make a more objective evaluation of the labels, without the previous iterations of the algorithm spoiling it. This solution was thought of because, in the previous version, when arriving at the last nodes, the algorithm would find a greater number of labels eliminated than at the first nodes.

From the figure 4, second and third images, we can see the difference between the last two implementations. In particular, it can be seen that node 10, which was previously maintained because the number of neighboring nodes with the same label and the number of nodes with a different label were equal, in this case, has lost its label, since of the three neighboring nodes, two have different labels. This may result in nodes losing or keeping their labels in these particular cases, compared to the previous implementation.

Label Propagation

The label propagation phase remained identical to that developed in ClassGraph. For this reason, it was decided to follow the ClassGraph pipeline as closely as possible. In addition to the use of long reads, the aim was to see whether performance could be improved by a new implementation of label propagation as was done previously, or to work on the classification of the binning tool. After seeing that new implementations of label propagation did not have the desired result, we proceeded to implementations of RefineLabel algorithm.

Experimental Setup

Evaluation parameters

The most commonly used parameters to evaluate classification tools are generally Recall and Accuracy. These two measures, together with the F1-Score and Pearson's correlation coefficient (PCC) were taken into account to study the performance of ClassGraph, compared to other binning tools, so for ClassGraph 2.0 we kept these parameters in order to have as accurate an evaluation as possible. In order to calculate these measures, it is necessary to know the actual taxonomic ids of the reads in the dataset used. The file that stores this information is also called the truth file or groud truth. This is compared with the results obtained by the classifier taking into account the desired taxonomic level. From this, 4 different possible binning results are identified:

- True Positive (TP) : the TaxId assigned by the classifier coincides or is a descendant of the correct one.
- False Negative (FN): the binning tool fails to perform any classification.
- Vague Positive (VP): Assignment of a Tax-Id that is correct, but more generic than the one requested.
- False Positive (FP): incorrect classification.

From these parameters it is therefore possible to calculate the evaluation measures mentioned above, as illustrated below. The Recall or Sensitivity represents the proportion of correctly classified reads in relation to the total number of reads.

$$Sensitivity = \frac{Numberofcorrectlyclassifiedreads}{Totalnumberofreads} = \frac{TP}{TP + VP + FN + FP}$$

Precision (PPV) is defined as the number of correctly classified reads out of the total number of True Positive and False Positive reads.

$$= \frac{Numberofcorrectlyclassifiedreads}{SumofTruePositiveandFalsePositivereads} = \frac{TP}{TP + FP}$$

F1-score relates Sensitivity and Precision, calculating the harmonic mean as follows.

$$F1 - score = \frac{2 * Sensitivity * PPV}{Sensitivity + PPV} = \frac{2 * TP}{2 * TP + VP + FN + 2 * FP}$$

Finally, the Pearson Correlation Coefficient (PCC) relates two variables X and Y, expressing a possible linearity relationship. Its value is between -1 and +1, where -1 corresponds to a negative perfect linear correlation, 0 to the absence of linear correlation and +1 to a positive perfect linear correlation. In the context of the evaluation of the binning tools, the two variables X and Y represent the species from the classifier and truth file results respectively.

$$PCC = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The symbols $\bar{x}$ and $\bar{y}$ denote the average of x and y respectively. Each of these four metrics was used to compare ClassGraph and the other binning tools in the experiments carried out. The assessment was made at the species level using the evaluate_calls.cc tool in: *https://github.com/DerrickWood/kraken2-experiment-code/tree/master/Utilities*.

Datasets

We evaluate our pipeline using four simulated datasets. The table 10 summarizes the characteristics of the datasets that have been analyzed.

Table 1: Summary of the datasets

Dataset	No. of species	Dataset size (GB)	No. of reads
Sim-8	8	1.06	128693
Sim-20	20	5.77	701450
Sim-50	50	13.12	1595637
Marine	5000	9.4	1644811
SRR7585901	—	9.4	763335

Three PacBio datasets were simulated using SimLoRD containing 8, 20, 50 species with the default PacBio erratic profiles in SimLoRD:

- Insertion = 0.11 (-pi)
- Deletion = 0.04 (-pd)
- Replacement = 0.01 (-ps)

These datasets are referred to as Sim-8, Sim-20 and Sim-50, respectively. Detailed dataset information is available in appendix .1, as the use of the -c parameter, which corresponds to the desired read coverage, is used to calculate the number of reads.

The Marine is a simulated dataset of short and long read shotgun metagenome data from samples at different locations on the seabed of a marine environment from the 2nd CAMI Challenge. Methodological improvements are difficult to evaluate due to the lack of a general standard for comparison. Critical Appraisal of Metagenome Interpretation (CAMI) is a new community-led initiative designed to help address these issues by aiming for an independent, comprehensive and unbiased evaluation of methods (5).

The real dataset is downloaded from (6). Of this dataset we do not have all the information available but only that stated by NCBI. Among the information, in addition to that in the table 10, we have the average length of the reads = 6566, the standard deviation = 5709,5. A taxonomy analysis is also available. We have at our disposal a taxonomy analysis that can be seen with a Krona graph in the figure 5 with which we can get an idea of the complexity of the dataset.

Kraken2 database

The first kraken2 database has a library that includes: archaea, bacteria, human, UniVec-Core and viral.

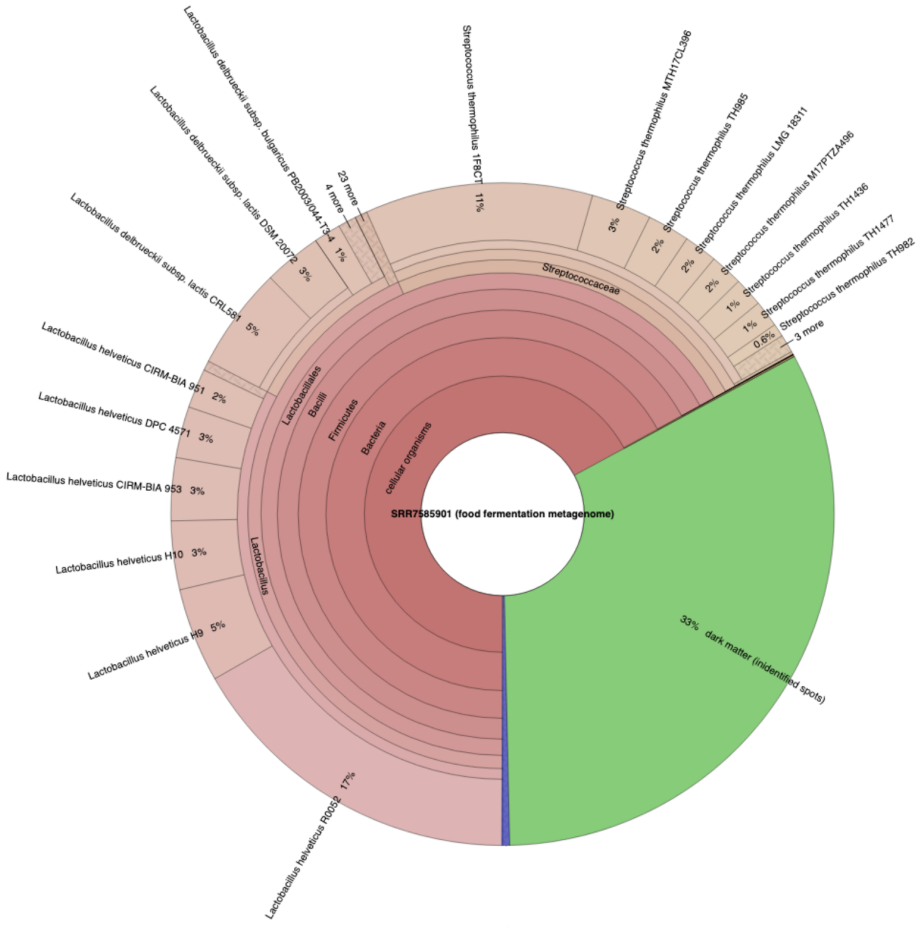


Figure 5: Taxonomy analysis of SRR7585901 dataset (7).

Strex Database

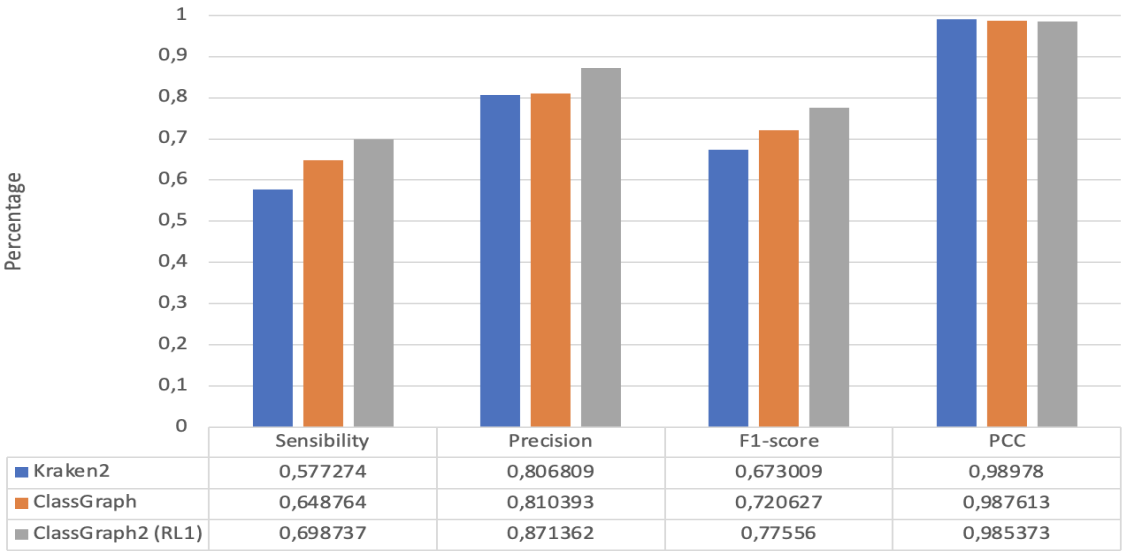
The Kraken2 database called strex was built using the procedure described in (2) was used. This procedure is called by the authors "Generation of data for strain exclusion experiments". Reference genomes and taxonomy were downloaded from the NCBI database. In particular, bacterial genomes were downloaded from (8) and virus genomes from (9). With regard to taxonomy, the files taxdump.tar.gz, containing the taxonomic tree of the data nodes in question, and gi-taxid-nucl.tar.gz, which associates the GenBank identifiers of the nucleotide records with their taxonomic ids, were downloaded. For each library (viruses and bacteria), only the complete genomes were then selected, excluding plasmids and the second and third chromosomes. This left a list containing one entry per genome, to which taxid information was added. From this list, a subset of taxids was chosen, of which there were two species and two sister subspecies in the database. The subset was then sorted by genus, species and strain and from this 40 strains were randomly extracted for bacteria and 10 for viruses, all belonging to different genera. At this point, the set of reference genomes was created by taking all those downloaded from NCBI and removing the 50 newly selected strains. The selected bacteria and viruses origin strains are listed in appendix .2.

Results and discussion

The amount of data and the memory required for metagenomic classification is very high, so the use of the Blade Computing cluster (10) is necessary to obtain results in an acceptable timeframe. The blade server installed at the Department of Computer Engineering (DEI) is configured to satisfy users' requests for computing resources in non-interactive or batch mode. It is possible to run on the cluster any 32/64 bit application or any program you have compiled. The various requests are taken over by the server management program, queued according to a priority system, and executed by the most suitable computing resource at that moment. Blade utilizes the Sun Grid Engine to manage the queue. This is a queuing system that allows a job to be executed according to specified requirements. The results of the calculation are saved in an output file.

ClassGraph 2.0 in Marine dataset

The first evaluations were made on the addition of the RefineLabel 1.0 algorithm to ClassGraph. As can be seen in the figure 6, this brought a good increase in performance in the Marine dataset. In particular, there was a good increase in sensitivity and precision, which led to an increase in the F1-Score. The PPC remained constant near its maximum, so in the next graphs it has been omitted to make them clearer. This



	Sensibility	Precision	F1-score	PCC
Kraken2	0,577274	0,806809	0,673009	0,98978
ClassGraph	0,648764	0,810393	0,720627	0,987613
ClassGraph2 (RL1)	0,698737	0,871362	0,77556	0,985373

Figure 6: Comparison of ClassGraph and ClassGraph 2.0 runs with a graph from the Marine dataset having $\frac{Arcs}{Nodes} = 2.5$ found with parameters k = 18 and m = 150 with the complete Kraken2 database.

suggests that in order to increase the performance of ClassGraph, it is not necessary to develop other LabelPropagation algorithms but to work on the labels already assigned by Kraken2. These experimental results show that there is a large number of labels incorrectly assigned by the starting tool.

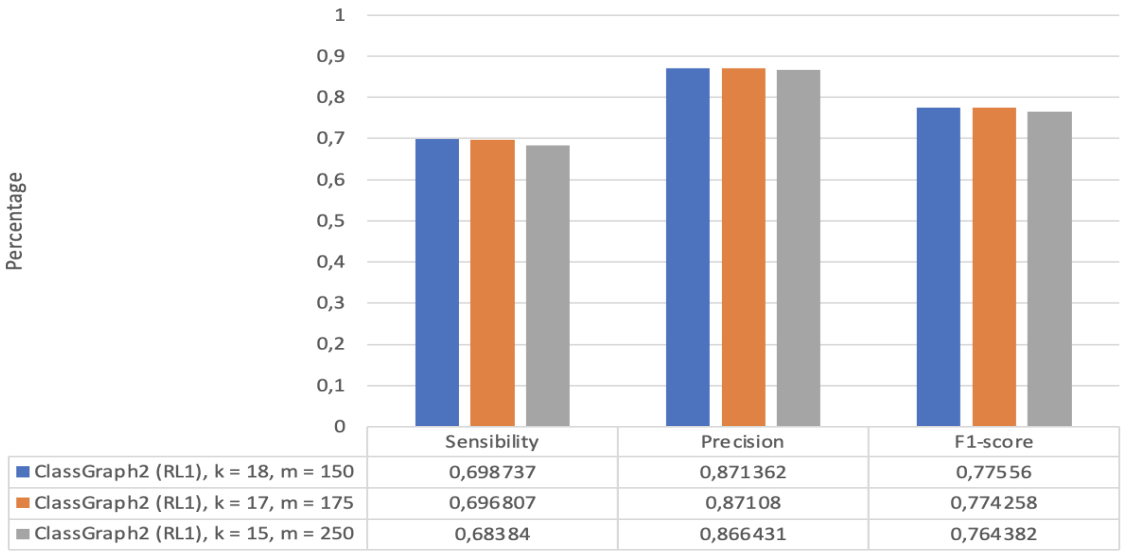
The first evaluation that has been made is on the parameters of minimap2 to find the graph to give as input to ClassGraph 2.0. In particular, the parameters of interest were -k and -m. Starting from the following preset option:

```
minimap2 -x ava-pb -k -m
(ava-pb: PacBio CLR all-vs-all overlap mapping (-Hk19 -Xw5 -e0 -m100))
```

which was explained earlier. By adding the parameters -k and -m to the command, they override the preset options, which are set to k = 19 and m = 100. It has been noted that as -k decreases, the execution times of minimap2 increase exponentially without making any significant changes to the results. Varying -m, on the other hand, brings important changes.

From figure 7 we can see that when the parameters -k and -m vary, keeping the $\frac{Arcs}{Nodes}$ constant, the performance remains the same at the expense of the computational costs, which increase as -k decreases. Through these results, it was understood that it was better to focus on $\frac{Arcs}{Nodes}$ rather than on -k. The same kind of reasoning has been done for different levels of connection of the graph. In particular, graphs with an $\frac{Arcs}{Nodes} = 12/16$ and an $\frac{Arcs}{Nodes} = 24/26$ were used. The results confirmed the described findings (Appendix .3).

As shown in Figure 8, there is a good increase in performance with increasing $\frac{Arcs}{Nodes}$. In particular, there is a clear increase between 2.5 and 16.6 $\frac{Arcs}{Nodes}$, after which the gap



	Sensibility	Precision	F1-score
ClassGraph2 (RL1), k = 18, m = 150	0,698737	0,871362	0,77556
ClassGraph2 (RL1), k = 17, m = 175	0,696807	0,87108	0,774258
ClassGraph2 (RL1), k = 15, m = 250	0,68384	0,866431	0,764382

Figure 7: ClassGraph 2.0 executions when the parameters -k and -m are changed while keeping constant $\frac{Arcs}{Nodes} = 2.5$ in the Marine dataset with the complete keaken2 database.

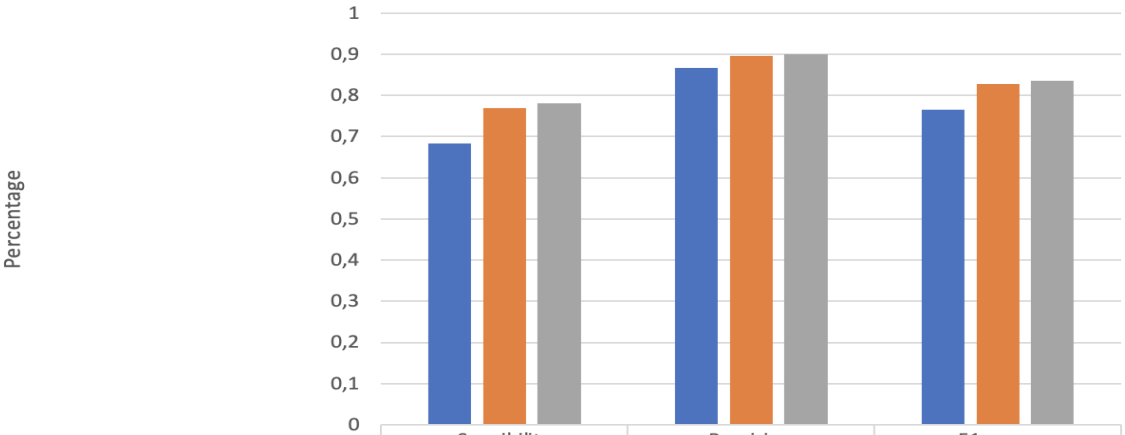
disappears. This suggests that above a certain threshold there will be no improvement but only an increase in computational cost. This reasoning has also been made for several -k. In particular, we used -k = 18, -k = 17 and -k = 15. From the results, we see no change in performance, so for the next tests, we left the -k of the preset option: -k = 19. The complete results can be found in 15.

After these evaluations, we proceeded with the testing of the other RefineLabel algorithms in Marine dataset using a relatively high $\frac{Arcs}{Nodes}$, which can be seen in figure 9.

It can be seen that the different implementations of RefineLabel increase the performance of the tool. In particular, it can be seen that from the first implementation there is a large increase in sensitivity and precision, and therefore also an increase in the F1-Score. The subsequent implementations further increase the advantage. This type of evaluation has also been done with less connected graphs. Graphs with $\frac{Arcs}{Nodes} = 2.5$ and $\frac{Arcs}{Nodes} = 12.5$ were used. The results are in agreement with those illustrated, although with slightly lower performance given the considerations made above. More information on this type of evaluation can be found in 16.

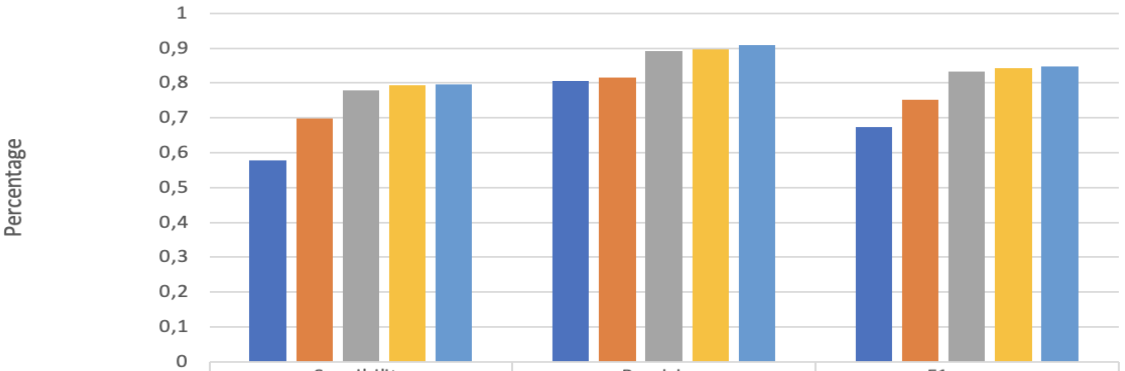
ClassGraph 2.0 in Simulated datasets

To confirm the increase in performance with the Marine dataset, several simulated datasets were constructed with increasing numbers of species and, therefore, complexity. These are the Sim-8, Sim-20, and Sim-50 datasets described in . These tests were carried out on the Kraken2 output with the complete database 10 and on the output with the Strex database 11.



	Sensibility	Precision	F1-score
ClassGraph2 (RL1), 2,5 archs/nodes	0,68384	0,866431	0,764382
ClassGraph2 (RL1), 16,6 archs/nodes	0,768064	0,896672	0,8274
ClassGraph2 (RL1), 26.5 archs/nodes	0,780833	0,899494	0,835974

Figure 8: Change in performance in relation to $\frac{Arcs}{Nodes}$ in the Marine dataset with the complete keaken2 database.



	Sensibility	Precision	F1-score
Kraken2	0,577274	0,806809	0,673009
ClassGraph	0,697363	0,814654	0,751459
ClassGraph2 (RL1)	0,77978	0,891683	0,831986
ClassGraph2 (RL2)	0,794224	0,897167	0,842563
ClassGraph2 (RL3)	0,795554	0,909551	0,848742

Figure 9: Evaluations of RefineLabel algorithms in marine datasets ($\frac{Arcs}{Nodes} = 25$) with the complete Keaken2 database.

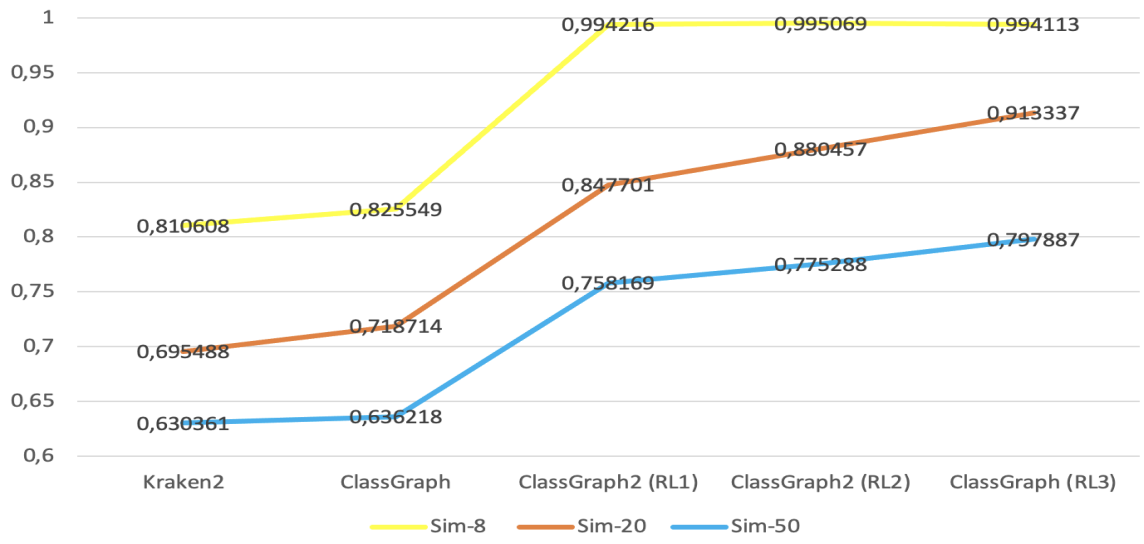


Figure 10: Evaluation of the F1-Score of simulated datasets using kraken2 output with complete database.

From these results, it can be clearly see the increase in performance compared to the starting tool, ClassGraph and the various implementations of RefineLabel algorithm. In particular, it can be seen that RefineLabel 1.0 is almost perfect for Sim-8. Even for more complex datasets, the increase in performance is clear, but decreases with the complexity of the dataset.

In figure 11 it can see that the same type of evaluation on the simulated datasets but with the Kraken2 Strex database.

The results are in agreement with those shown above. The Strex database, as mentioned in Section , is composed of species of bacteria and viruses. The analyzed datasets are, in turn, composed of long reads of species of bacteria and viruses. This caused the evaluations of the Kraken2 binning tool to be better than using the complete database, which also includes other libraries. The complete results can be found in 17.

Normalization at species level

After these evaluations, we wanted to go further. As the initial binning tool Kraken2 also classifies at levels below species level and ground truths are at species level, many labels would be incorrectly removed. As soon as the RefineLabel algorithms detect a divergence between labels, they are removed. Unfortunately, if a vertex labeled at the species level is connected to a vertex labeled at the subspecies level but of the same species, these labels are considered different by the algorithm, but this leads to an error. To analyze this problem, in the Kraken2 output, all classifications have been changed from subspecies to species by modifying TaxId. Classifications at higher levels were removed.

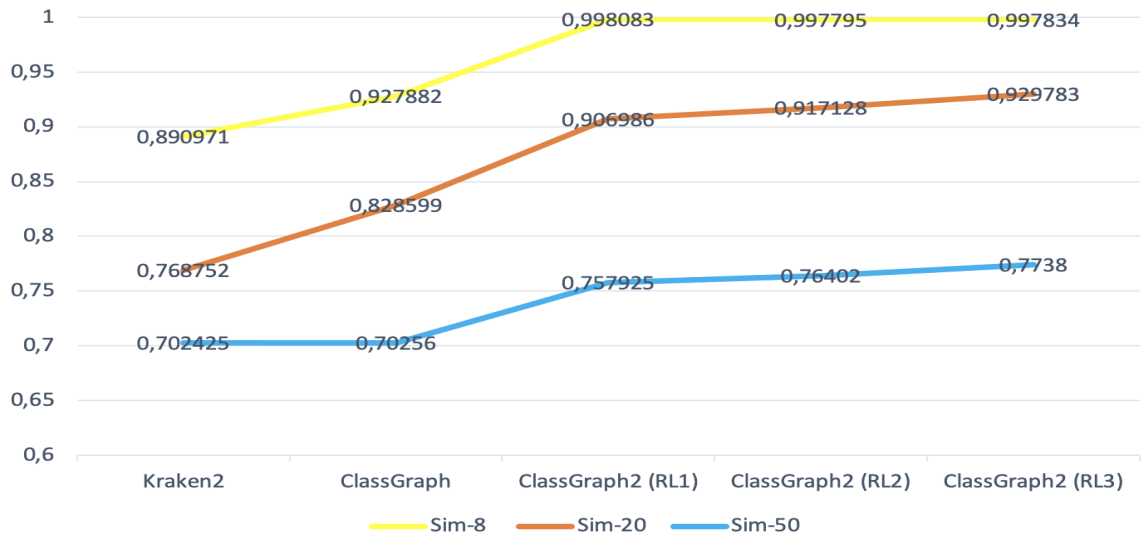


Figure 11: Evaluation of the F1-Score of simulated datasets using kraken2 output with the Strex database.

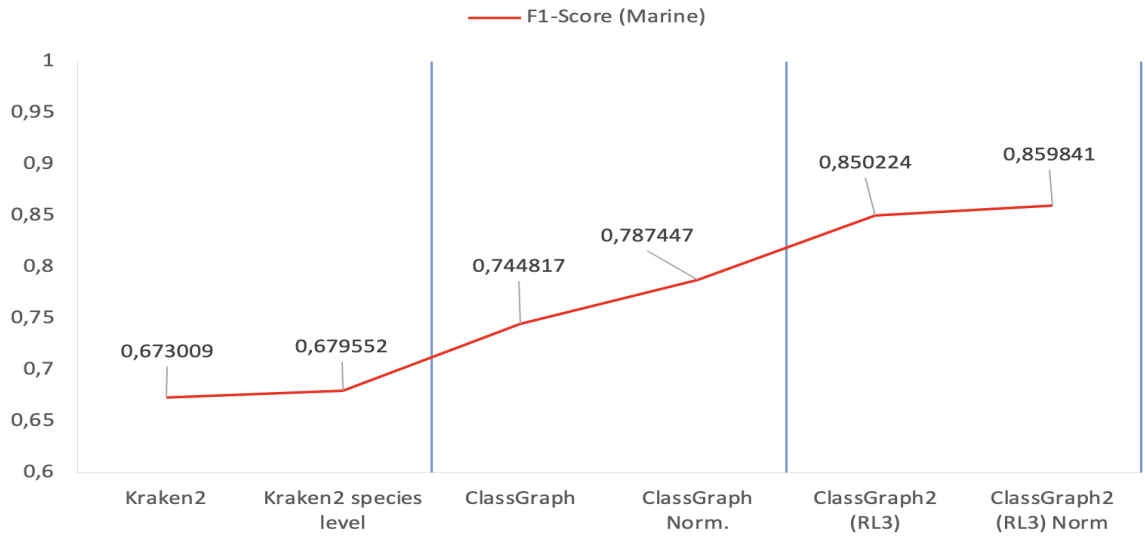


Figure 12: Normalization at the species level in Marine dataset

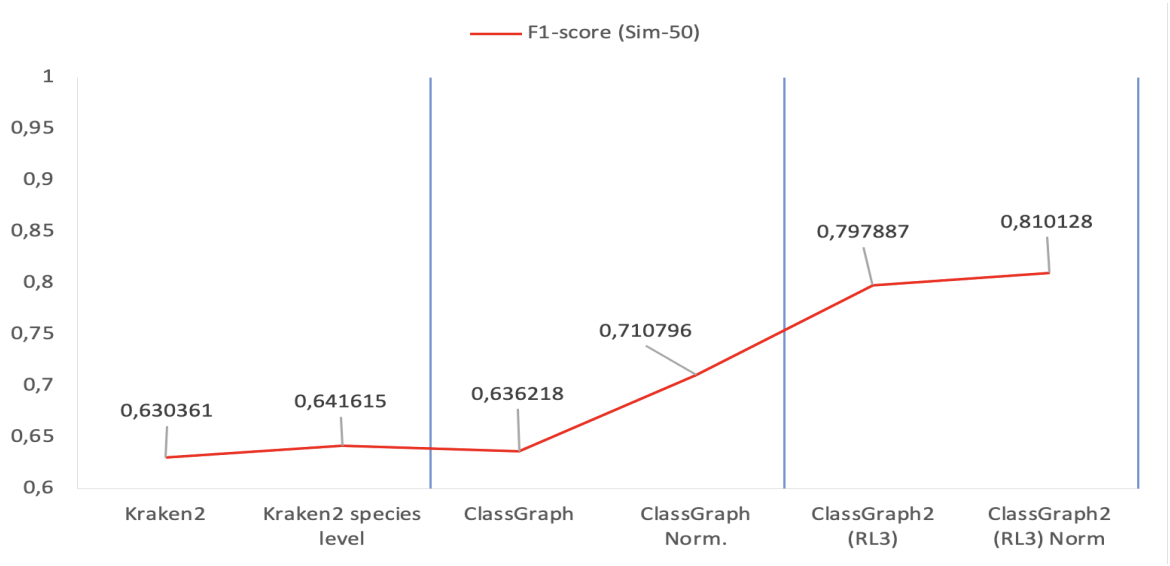


Figure 13: Normalization at the species level in Sim-50 dataset

The figures 12 and 13 confirms these assumptions. It can be seen that for both of the more complex datasets analyzed, the Sim-50 and the Marine, after normalization at the ClassGraph 2.0 species level (Norm.), performance increases considerably. The first increase, with ClassGraph, is given by the elimination of labels at levels higher than species level, thanks to which the LabelPropagation algorithm has more possibilities to expand the labels to unlabeled nodes. ClassGraph 2.0 further increases performance compared to previous results in which the normalization of labels did not appear. Extending this discussion, it was observed that after normalization at species level, all labels classified as vague positive are eliminated. These labels after the execution of ClassGraph 2.0 increase the number of true positives, false positives, and false negatives, as can be seen in the tables 2 and 3.

Table 2: Variation true positive, false positive, false negative and vague positive in Sim-50 dataset.

	True positive	**False positive**	**False negative**	**Vague positive**
kraken2	910330	382313	118551	184443
kraken2 Norm.	910330	331653	353654	0
ClassGraph	951195	443321	8540	192581
ClassGraph Norm.	1130129	454131	11377	0
ClassGraph 2.0 (RL3)	1228590	255381	26013	85653
ClassGraph 2.0 (RL3) Norm.	1284914	291575	19148	0

Abundance on real dataset

The last analysis was to compare the abundance of species in the SRR7585901 dataset. This dataset can be downloaded from NCBI (6). The dataset consists of 763335 long reads. The comparison was made between the Kraken2 classification result with a confidence score = 0.01, which resulted in a percentage of unclassified reads of 27.05%.

Table 3: Variation true positive, false positive, false negative and vague positive in Marine dataset.

	True positive	False positive	False negative	Vague positive
kraken2	663188	158801	195331	131507
kraken2 Norm.	663188	139164	347135	0
ClassGraph	789069	181241	29873	148334
ClassGraph Norm.	881847	209257	57555	0
ClassGraph 2.0 (RL3)	901528	70502	69302	107322
ClassGraph 2.0 (RL3) Norm.	953708	115872	79175	0

So the number of unclassified reads before the execution of ClassGraph 2.0 (RL3) was 206485. After the execution, thanks to the high connectivity of the graph of $\frac{Arcs}{Nodes} = 60.3$, the unclassified reads are 125867, 16.48%. It was therefore possible to reach parts of the graph that previously could not be reached thanks to LabelPropagation. In particular, the number of iterations of LabelPropagation was 5 in total. The variation in abundance of the species can be seen in the table 4. The percentage values are calculated on the total reads.

Table 4: Variation in species abundance in the SRR7585901 dataset.

TaxId	Organism name	Rank	Previous abundance	Next abundance
0	Unclassified		27.05%	16.48%
1308	Streptococcus thermophilus	Species	19.51%	27.52%
1578	Lactobacillus	Genus	22.22%	25.34%
1587	Lactobacillus helveticus	Species	11.13%	13.07%
1584	Lactobacillus delbrueckii	Species	8.99%	10.57%
1301	Streptococcus	Genus	5.36%	3.00%
9606	Homo sapiens	Species	1.2%	0.71%
33958	Lactobacillaceae	Family	0.55%	0.60%
131567	cellular organisms	No rank	0.63%	0.33%
1224	Proteobacteria	Phylum	0.39%	0.36%

To find the values shown, we worked on the output of Kraken2 and the output of ClassGraph 2.0, counting the number of nodes with the same label. From the results we can see that for the most abundant species the pre-census of abundance increases, while for the less abundant species it decreases. Unfortunately we do not have ground truth of real datasets, but for the considerations made for the simulated datasets, it is reasonable to think that the increase of abundances for the species most present in the dataset corresponds to an increase in the accuracy of the classification. It has been experimentally verified that the increase in abundance does not occur for loosely connected graphs, as is reasonable to assume, and for higher initial labelling rates. That is, after running ClassGraph 2.0 it is not possible to arrive at isolated parts of the graph.

Time and memory cost

The figure 14 gives an idea of the computational costs. We can see that for all the simulated datasets, the time required to run Kraken2 and that of Classgraph and

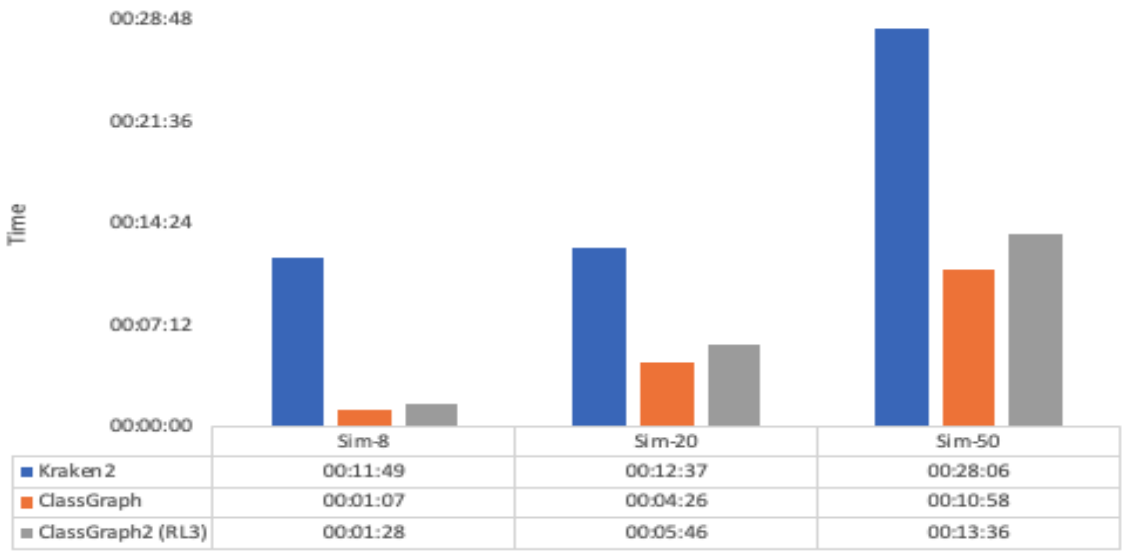


Figure 14: Time required for executions

ClassGraph 2.0 are of the same order of size. For example, in Sim-50 the addition of the LabelPropagation algorithm increases the time cost by only 00:10:58. On the other hand, adding both the LabelPropagation and RefineLabel algorithms increases the temporal cost by 00:13:36. However, against this increase, there is a significant increase in the F1-Score as previously described.

Table 5: Memory required for execution in the complete Kraken2 database.

	Sim-8	**Sim-20**	**Sim-50**
Kraken2	47.45 GB	47.45 GB	47.45 GB
ClassGraph	2.25 GB	11.15 GB	28.56 GB
ClassGraph 2.0 (RL3)	2.72 GB	13.22 GB	33.91 GB

With regard to the required memory , it can be seen from the table 5 that Kraken2 makes great use of memory for the entire database. However, when using ClassGraph and ClassGraph 2.0, the memory required depends on the size of the dataset used. Between ClassGraph and ClassGraph 2.0 there is a slight increase in memory for the RefineLabel functions.

All time and memory costs for both the full Kraken2 and Strex databases are listed in the Appendix .3.

Conclusion

The purpose of this thesis was to improve the performance of kraken2, a metagenomic classifier. To do this, we started from ClassGraph, keeping its LabelPropagation algorithm, but working on long reads. This in itself had already increased the performance in terms of Sensitivity, Precision, F1-score and PCC. After the inclusion of the RefineLabel

algorithms, these have increased significantly. This showed that part of the Kraken2 classification is wrong. It is therefore necessary to delete the labels assigned by him. All the experiments carried out on the different datasets gave concordant results. In particular, it has been discovered that by increasing the connectivity of the graph and varying the parameters of minimap2, the LabelPropagatio and RefineLabel algorithms work better, managing to arrive at parts of the graph that would otherwise remain isolated. In the future, it would be interesting to develop a new RefineLabel algorithm that takes into account the weights of the arcs and no longer their number, in order to have an increasingly targeted elimination of labels. However, from the results with these datasets it has been seen that if the starting performances are high, it will be increasingly difficult to improve them. You will therefore need very complex datasets, where the initial classification is very difficult, perhaps a real dataset available to the ground truth. Another future work could be to try to give ClassGraph 2.0 as input the result of other classifiers such as Kraken, Centrifuge, CLARK and so on. Each classifier works with different algorithms, so the inputs to ClassGraph 2.0 would be completely different from those used in this thesis work.

.1 Dataset Information

In tables 6, 7, 8 you can find all the information regarding the simulated datasets.

Table 6: Sim-8: 1.06GB, 128693 long reads.

TaxId	Species name	Coverage	No. reads
749927	Amycolatopsis mediterranei U32	25	31168
203119	Acetivibrio thermocellus ATCC-27405	31	14510
438753	Azorhizobium caulinodans ORS 571	20	16349
264462	Bdellovibrio bacteriovorus HD100	33	9675
442563	Bifidobacterium animalis subsp lactis AD011	40	7300
1045858	Brachyspira intermedia PWS	150	8057
1265757	Candidatus Pelagibacter ubique HTCC1013	80	10617
861360	Glutamicibacter arilaitensis Re117	65	31017

.2 Genomes used for the strain exclusion experiment

The following tables (10 and 9) list the strains obtained during the generation of the strain exclusion data explained in section .

.3 Tables with all available data.

In figures 15, 16, 17 you can find all the information regarding all the results obtained.

Table 7: Sim-20: 5.77GB, 701450 long reads.

TaxId	Species name	Coverage	No. reads
203119	Acetivibrio thermocellus ATCC-27405	31	14510
749927	Amycolatopsis mediterranei U32	25	31168
438753	Azorhizobium caulinodans ORS 571	20	16349
264462	Bdellovibrio bacteriovorus HD100	33	9675
442563	Bifidobacterium animalis subsp lactis AD011	40	7300
1045858	Brachyspira intermedia PWS	150	8057
1265757	Candidatus Pelagibacter ubique HTCC1013	80	10617
945711	Corynebacterium ulcerans 809	20	10056
525280	Erysipelothrix rhusiopathiae ATCC 19414	85	8508
1088879	Francisella tularensis subsp. novicida FSC159	150	34874
861360	Glutamicibacter arilaitensis Re117	65	31017
2243	Halobacterium sp.	65	21617
1195469	Mycobacterium tuberculosis variant bovis BCG str. Sweden	80	39740
105559	Nitrosococcus watsonii C-113	95	39029
61624	Paenibacillus mucilaginosus strain G78 Scaffold1	90	94013
314280	Photobacterium profundum 3TCK	45	64041
1105097	Rickettsia prowazekii str. Dachau	100	13507
292459	Symbiobacterium thermophilum IAM 14863	250	108579
604354	Thermococcus sibiricus MM 739	210	47207
280	Xanthobacter autotrophicus	150	91585

k = 18		k = 17		k = 15	
m = 150		m = 175		m = 250	
Arch/Node = 2.5		Arch/Node = 2.6		Arch/Node = 2.7	
ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)
Sens: 0.648764 Prec: 0.810393 f1: 0.720627 PCC: 0.987613 Time: 00:01:03 Memory: 3.47 GB	Sens: 0.698737 Prec: 0.871362 f1: 0.775560 PCC: 0.985373 Time: 00:01:27 Memory: 3.83 GB	Sens: 0.647404 Prec: 0.810364 f1: 0.719775 PCC: 0.987632 Time: 00:01:36 Memory: 3.72 GB	Sens: 0.696807 Prec: 0.871080 f1: 0.774258 PCC: 0.985423 Time: 00:02:01 Memory: 4.11 GB	Sens: 0.639308 Prec: 0.810208 f1: 0.714684 PCC: 0.987632 Time: 00:02:43 Memory: 3.94 GB	Sens: 0.683840 Prec: 0.866431 f1: 0.764382 PCC: 0.985423 Time: 00:03:27 Memory: 4.46 GB
k = 18		k = 17		k = 15	
m = 75		m = 100		m = 137	
Arch/Node = 13.6		Arch/Node = 12.4		Arch/Node = 16.6	
ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)
Sens: 0.690354 Prec: 0.813565 f1: 0.746913 PCC: 0.989380 Time: 00:07:30 Memory: 14.51 GB	Sens: 0.772089 Prec: 0.896774 f1: 0.829774 PCC: 0.988093 Time: 00:08:55 Memory: 16.68 GB	Sens: 0.687033 Prec: 0.813213 f1: 0.744817 PCC: 0.989272 Time: 00:06:46 Memory: 13.38 GB	Sens: 0.766074 Prec: 0.894730 f1: 0.825419 PCC: 0.988084 Time: 00:08:06 Memory: 15.09 GB	Sens: 0.688821 Prec: 0.813815 f1: 0.746119 PCC: 0.989345 Time: 00:13:58 Memory: 17.87 GB	Sens: 0.768064 Prec: 0.896672 f1: 0.827400 PCC: 0.988142 Time: 00:17:01 Memory: 20.65 GB
k = 18		k = 17		k = 15	
m = 36		m = 60		m = 100	
Arch/Node = 24.7		Arch/Node = 25		Arch/Node = 26.5	
ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)	ClassGraph	ClassGraph2 (RL1)
Sens: 0.696590 Prec: 0.803930 f1: 0.746420 PCC: 0.989569 Time: 00:15:44 Memory: 25.61 GB	Sens: 0.658492 Prec: 0.719633 f1: 0.687706 PCC: 0.981031 Time: 00:18:26 Memory: 31.54 GB	Sens: 0.697363 Prec: 0.814654 f1: 0.751459 PCC: 0.989609 Time: 00:19:00 Memory: 26.18 GB	Sens: 0.779780 Prec: 0.891683 f1: 0.831986 PCC: 0.987807 Time: 00:21:23 Memory: 30.29 GB	Sens: 0.696503 Prec: 0.814994 f1: 0.751104 PCC: 0.989588 Time: 00:29:13 Memory: 28.08 GB	Sens: 0.780833 Prec: 0.899494 f1: 0.835974 PCC: 0.988218 Time: 00:33:07 Memory: 32.35 GB

Figure 15: Data on the evaluation of the $\frac{Arcs}{Nodes}$ in the Marine dataset.

Table 8: Sim-50: 13.12GB, 1595637 long reads.

TaxId	Species name	Coverage	No. reads
203119	Acetivibrio thermocellus ATCC-27405	31	14510
749927	Amycolatopsis mediterranei U32	25	31168
438753	Azorhizobium caulinodans ORS 571	20	16349
2026187	Bacillus pacificus	20	13194
264462	Bdellovibrio bacteriovorus HD100	33	9675
367928	Bifidobacterium adolescentis ATCC 15703	40	10179
442563	Bifidobacterium animalis subsp lactis AD011	40	7300
1045858	Brachyspira intermedia PWS	150	8057
237561	Candida albicans SC5314	70	121763
1265757	Candidatus Pelagibacter ubique HTCC1013	80	10617
272943	Cereibacter sphaeroides 2.4.1	100	56059
290402	Clostridium beijerinckii NCIMB 8052	30	21924
212717	Clostridium tetani E88	130	45492
945711	Corynebacterium ulcerans 809	20	10056
243230	Deinococcus radiodurans R1	90	35997
254945	Ehrlichia ruminantium str. Welgevonden	190	35088
525280	Erysipelothrix rhusiopathiae ATCC 19414	85	8508
411485	Faecalibacterium prausnitzii M21/2	20	7617
1163730	Fervidicoccus fontis Kam940	120	19279
1088879	Francisella tularensis subsp. novicida FSC159	150	34874
393480	Fusobacterium nucleatum subsp. polymorphum ATCC 10953	55	16355
861360	Glutamicibacter arilaitensis Re117	65	31017
888828	Haemophilus parainfluenzae ATCC 33392	75	19408
2243	Halobacterium sp.	65	21617
731	Histophilus somni	95	26193
324831	Lactobacillus gasseri ATCC 33323 = JCM 1131	225	51910
363253	Lawsonia intracellularis PHE/MN1-00	50	10467
334390	Limosilactobacillus fermentum IFO 3956	125	31949
1006006	Metallosphaera cuprina Ar-4	30	6724
420247	Methanobrevibacter smithii ATCC 35061	40	9027
1434109	Methanosarcina barkeri str. Wiesmoor	60	35858
465515	Micrococcus luteus NCTC 2665	80	24368
1195469	Mycobacterium tuberculosis variant bovis BCG str. Sweden	80	39740
710128	Mycoplasma gallisepticum str. R(high)	20	2465
105559	Nitrosococcus watsonii C-113	95	39029
61624	Paenibacillus mucilaginosus strain G78 Scaffold1	90	94013
314280	Photobacterium profundum 3TCK	45	64041
242619	Porphyromonas gingivalis W83	90	25687
1122981	Prevotella corporis DSM 18810 = JCM 8529	200	68207
272943	Rickettsia prowazekii str. Dachau	100	56059
1105097	Rickettsia rickettsii str. Iowa	100	13507
585394	Roseburia hominis A2-183	80	34998
559292	Saccharomyces cerevisiae S288C	60	88836
411466	Schaalia odontolytica ATCC 17982	60	17493
680198	Streptomyces scabiei 87.22	40	49440
292459	Symbiobacterium thermophilum IAM 14863	250	108579
604354	Thermococcus sibiricus MM 739	210	47207
595537	Variovorax paradoxus EPS	30	23931
1298595	Veillonella rogosae JCM 15642	20	5327
280	Xanthobacter autotrophicus	150	91585

Table 9: Viruses genomes used as origin stains in the strain exclusion experiment.

TaxId	Scientific name
1070413	Human papillomavirus 140
41856	Hepatitis C virus genotype 1
1087109	Canis familiaris papillomavirus 10
981431	Pseudomonas phage PAK P3
1156769	Porcine kobuvirus
57579	Adeno-associated virus - 4
12524	Junonia coenia densovirus
11801	Moloney murine leukemia virus
89623	Snow goose hepatitis B virus
1458710	Mycobacterium phage Badfish

Dataset Marine 1644811 long reads	Arch/Node = 2.6, k = 17, m = 100	Arch/Node = 12.4, k= 17, m= 175	Arch/Node = 25, k = 17, m = 60
DB Complete	Sens: 0.577274 Prec: 0.806809 f1: 0.673009 PCC: 0.989780 Time: 00:12:31 Memory: 47.43 GB	Sens: 0.577274 Prec: 0.806809 f1: 0.673009 PCC: 0.989780 Time: 00:12:31 Memory: 47.43 GB	Sens: 0.577274 Prec: 0.806809 f1: 0.673009 PCC: 0.989780 Time: 00:12:31 Memory: 47.43 GB
ClassGraph	Sens: 0.647404 Prec: 0.810364 f1: 0.719775 PCC: 0.987632 Time: 00:01:26 Memory: 3.72 GB	Sens: 0.687033 Prec: 0.813213 f1: 0.744817 PCC: 0.989272 Time: 00:04:46 Memory: 13.38 GB	Sens: 0.697363 Prec: 0.814654 f1: 0.751459 PCC: 0.989609 Time: 00:14:00 Memory: 26.18 GB
ClassGraph2 (RL1)	Sens: 0.696807 Prec: 0.871080 f1: 0.774258 PCC: 0.985423 Time: 00:01:51 Memory: 4.11 GB	Sens: 0.766074 Prec: 0.894730 f1: 0.825419 PCC: 0.988084 Time: 00:05:06 Memory: 15.09 GB	Sens: 0.779780 Prec: 0.891683 f1: 0.831986 PCC: 0.987807 Time: 00:16:23 Memory: 30.29 GB
ClassGraph2 (RL2)	Sens: 0.709618 Prec: 0.866267 f1: 0.780157 PCC: 0.986800 Time: 00:01:32 Memory: 3.95 GB	Sens: 0.780523 Prec: 0.896123 f1: 0.834338 PCC: 0.989851 Time: 00:05:48 Memory: 14.67 GB	Sens: 0.794224 Prec: 0.897167 f1: 0.842563 PCC: 0.989554 Time: 00:18:01 Memory: 28.98 GB
ClassGraph2 (RL3)	Sens: 0.708488 Prec: 0.892730 f1: 0.790009 PCC: 0.987310 Time: 00:01:22 Memory: 3.97 GB	Sens: 0.784856 Prec: 0.927469 f1: 0.850224 PCC: 0.990451 Time: 00:05:00 Memory: 14.70 GB	Sens: 0.795554 Prec: 0.909551 f1: 0.848742 PCC: 0.989793 Time: 00:17:27 Memory: 29.01 GB

Figure 16: Evaluations of marine datasets.

Table 10: Bacteria genomes used as origin stains in the strain exclusion experiment.

TaxId	Name
706191	Pantoea ananatis LMG 20103
698969	Corynebacterium diphtheriae HC03
1105098	Rickettsia prowazekii str. GvV257
300852	Thermus thermophilus HB8
759913	Streptococcus dysgalactiae subsp. equisimilis AC-2713
401614	Francisella tularensis subsp. novicida U112
272559	Bacteroides fragilis NCTC 9343
1096995	Acinetobacter baumannii BJAB07104
882096	Listeria monocytogenes SLCC5850
863638	Clostridium acetobutylicum EA 2018
366649	Xanthomonas citri pv. fuscans
1117943	Sinorhizobium fredii HH103
1173064	Anaplasma phagocytophilum str. JM
354242	Campylobacter jejuni subsp. jejuni 81-176
1161918	Brachyspira pilosicoli WesB
1244085	Klebsiella pneumoniae CG43
936153	Enterococcus faecalis 62
591020	Shigella flexneri 2002017
243276	Treponema pallidum subsp. pallidum str. Nichols
374930	Haemophilus influenzae PittEE
1042876	Pseudomonas putida S16
395492	Rhizobium leguminosarum bv. trifolii WSM2304
909420	Neisseria meningitidis H44/76
1392476	Staphylococcus aureus subsp. aureus 6850
257310	Bordetella bronchiseptica RB50
336982	Mycobacterium tuberculosis F11
644042	Lactobacillus plantarum JDM1
138677	Chlamydia pneumoniae J138
402882	Shewanella baltica OS185
634997	Mycoplasma hyorhinis DBS 1050
1053692	Methanococcus maripaludis X1
224326	Borreliella burgdorferi B31
592021	Bacillus anthracis str. A0248
573059	Desulfovibrio vulgaris RCH1
1116391	Paenibacillus mucilaginosus 3016
434271	Actinobacillus pleuropneumoniae serovar 3 str. JL03
956149	Cronobacter sakazakii SP291
290847	Helicobacter pylori 51
386656	Yersinia pestis Pestoides F
1300259	Alteromonas mediterranea UM4b

	Sim8: 128693 reads, 1.06 GB, m = 50, 31.7 arch/node		**Sim20**: 701450 reads, 5.77 GB, m = 150, 28.9 arch/node		**Sim50**: 1595637 reads, 13.12 GB, m = 135, 32 arch/node	
	K2_DB-Complete 2.59% unclassified	K2_DB-strex 6.78% unclassified	K2_DB-Complete 5,34% unclassified	K2_DB-strex 14.06% unclassified	K2_DB-Complete 7.43% unclassified	K2_DB-strex 22.58% unclassified
Kraken2	Sens: 0.765628 Prec: 0.861202 f1: 0., 810608 PCC: 0.996426 Time: 00:11:49 Memory: 47.45 GB	Sens: 0.848313 Prec: 0.938146 f1: 0.890971 PCC: 0.999249 Time: 00:04:03 Memory: 9.75 GB	Sens: 0.629978 Prec: 0.776203 f1: 0.695488 PCC: 0.923726 Time: 00:12:37 Memory: 47.47 GB	Sens: 0.686822 Prec: 0.872876 f1: 0.768752 PCC: 0.939344 Time: 00:09:49 Memory: 9.80 GB	Sens: 0.570512 Prec: 0.704239 f1: 0.630361 PCC: 0.946056 Time: 00:28:06 Memory: 47.53 GB	Sens: 0.604992 Prec: 0.837267 f1: 0.702425 PCC: 0.956633 Time: 00:22:15 Memory: 9.85 GB
ClassGraph	Sens: 0.789763 Prec: 0.864731 f1: 0.825549 PCC: 0.995107 Time: 00:01:07 Memory: 2.25 GB	Sens: 0.913927 Prec: 0.942270 f1: 0.927882 PCC: 0.999000 Time: 00:01:04 Memory: 2.32 GB	Sens: 0.666136 Prec: 0.780303 f1: 0.718714 PCC: 0.933724 Time: 00:04:26 Memory: 11.15 GB	Sens: 0.790504 Prec: 0.870551 f1: 0.828599 PCC: 0.969923 Time: 00:03:45 Memory: 11.53 GB	Sens: 0.596122 Prec: 0.682097 f1: 0.636218 PCC: 0.952594 Time: 00:10:58 Memory: 28.56 GB	Sens: 0.677481 Prec: 0.729568 f1: 0.702560 PCC: 0.978374 Time: 00:12:07 Memory: 30.10 GB
ClassGraph2 (RL1)	Sens: 0.993045 Prec: 0.995389 f1: 0.994216 PCC: 0.999967 Time: 00:01:18 Memory: 2.85 GB	Sens: 0.997273 Prec: 0.998895 f1: 0.998083 PCC: 0.999999 Time: 00:01:16 Memory: 2.86 GB	Sens: 0.800020 Prec: 0.901425 f1: 0.847701 PCC: 0.943355 Time: 00:04:42 Memory: 14.10 GB	Sens: 0.873367 Prec: 0.943298 f1: 0.906986 PCC: 0.979248 Time: 00:04:13 Memory: 13.44 GB	Sens: 0.728687 Prec: 0.790138 f1: 0.758169 PCC: 0.959146 Time: 00:12:01 Memory: 36.35 GB	Sens: 0.737146 Prec: 0.779909 f1: 0.757925 PCC: 0.984002 Time: 00:13:32 Memory: 35.11 GB
ClassGraph2 (RL2)	Sens: 0.994227 Prec: 0.995914 f1: 0.995069 PCC: 0.999974 Time: 00:01:27 Memory: 2.72 GB	Sens: 0.996985 Prec: 0.998607 f1: 0.997795 PCC: 0.999995 Time: 00:01:25 Memory: 2.72 GB	Sens: 0.834902 Prec: 0.931271 f1: 0.880457 PCC: 0.964413 Time: 00:05:00 Memory: 13.18 GB	Sens: 0.884707 Prec: 0.952014 f1: 0.917128 PCC: 0.980668 Time: 00:07:48 Memory: 12.68 GB	Sens: 0.747205 Prec: 0.805565 f1: 0.775288 PCC: 0.970929 Time: 00:13:13 Memory: 33.78 GB	Sens: 0.743672 Prec: 0.785513 f1: 0.764020 PCC: 0.985014 Time: 00:15:31 Memory: 33.25 GB
ClassGraph2 (RL3)	Sens: 0.993279 Prec: 0.994949 f1: 0.994113 PCC: 0.999963 Time: 00:01:28 Memory: 2.72 GB	Sens: 0.997024 Prec: 0.998646 f1: 0.997834 PCC: 0.999995 Time: 00:01:26 Memory: 2.72 GB	Sens: 0.869446 Prec: 0.961895 f1: 0.913337 PCC: 0.977861 Time: 00:05:46 Memory: 13.22 GB	Sens: 0.895993 Prec: 0.966223 f1: 0.929783 PCC: 0.981626 Time: 00:05:37 Memory: 12.70 GB	Sens: 0.769968 Prec: 0.827907 f1: 0.797887 PCC: 0.983608 Time: 00:13:36 Memory: 33.91 GB	Sens: 0.750408 Prec: 0.798699 f1: 0.773800 PCC: 0.985730 Time: 00:14:09 Memory: 33.30 G

Figure 17: Evaluations of simulated datasets.

Bibliography

[1] M. C. M. Cavattoni, "Boosting metagenomic classification with reads overlap graph," 2020.

[2] D. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with kraken 2," *Genome Biology*, vol. 20, 11 2019.

[3] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, vol. 32, pp. 2103–2110, 03 2016.

[4] M. O. Pollard, D. Gurdasani, A. J. Mentzer, T. Porter, and M. S. Sandhu, "Long reads: their purpose and place," *Human Molecular Genetics*, vol. 27, pp. R234–R241, 05 2018.

[5] "Critical assessment of metagenome interpretation." https://data.cami-challenge.org/. Accessed: 2022-03-29.

[6] "Ncbi - srr7585901." https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR9328980. Accessed: 2022-03-24.

[7] "Srr7585901krona." https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR7585901&krona=on. Accessed: 2022-07-04.

[8] "Bacterial genome download." ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/Bacteria/all.fna.tar.gz. Accessed: 2022-01-21.

[9] "Virus genome download." ftp://ftp.ncbi.nlm.nih.gov/genomes/Viruses/all.fna.tar.gz. Accessed: 2022-01-21.

[10] "bladecluster." https://www.dei.unipd.it/bladecluster. Accessed: 2021-12-05.