SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1571

De Novo Metagenome Assembly Using Read Clustering

Luka Škugor

Zagreb, lipanj 2018.

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
MASTER THESIS COMMITTEE

Zagreb, 2 March 2018

MASTER THESIS ASSIGNMENT No. 1571

Student: **Luka Škugor (0036478527)**
Study: Computing
Profile: Computer Science

Title: **De Novo Metagenome Assembly Using Read Clustering**

Description:

Accurate and contiguous de novo genome assemblies are now possible owing to third generation of sequencing which produces much longer reads than its predecessors. The OLC paradigm, short for overlap-layout-consensus, is best tailored and mostly used for this task. The next logical step is to expand de novo assembly to metagenomes, i.e. de novo assembling samples containing DNA sequences of multiple different species. The goal of this work is to design and develop a tool which will cluster reads by present species and assemble each cluster separately using publicly available OLC assemblers. Clustering should be done using most appropriate machine learning algorithms. The solution should be appropriated for parallel architectures and implemented in C++ or Python. The source code has to be documented using comments and should follow the Google C++ or Python Style Guide when possible. The complete application should be hosted on GitHub under an OSI-approved license.

Issue date: 16 March 2018
Submission date: 29 June 2018

Mentor:
Associate Professor Mile Šikić, PhD

Committee Chair:
Full Professor Siniša Srbljić, PhD

Committee Secretary:
Assistant Professor Tomislav Hrkać, PhD

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 2. ožujka 2018.

DIPLOMSKI ZADATAK br. 1571

Pristupnik: **Luka Škugor (0036478527)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **De novo sastavljanje metagenoma koristeći metode za grupiranje očitanja**

Opis zadatka:

De novo sastavljanje genoma s malom stopom fragmentiranosti donekle je moguće zahvaljujući trećoj generaciji sekvenciranja koja proizvodi puno duža očitanja u odnosu na prethodne generacije. Najprikladnija i uz to najkorištenija metoda sastavljanja genoma je OLC paradigma, što je skraćeno od preklapanje-razmještaj-konsenzus. Sljedeći logičan korak u de novo sastavljanju genoma je proširenje na metagenome, tj. sastavljanje uzoraka koji sadrže DNA molekule više različitih organizama. Tema ovog rada je dizajn i razvoj alata koji će grupirati očitanja po prisutnim organizmima te sastaviti svaki skup zasebno pomoću već postojećih OLC asemblera. Za grupiranje očitanja treba isprobati nekoliko različitih metoda strojnog učenja te odabrati najbolju. Rješenje mora biti napisano u jeziku C++ ili Python. Programski kod je potrebno komentirati i pri pisanju pratiti stil opisan u Googleovom C++ ili Python vodiču. Kompletnu aplikaciju postaviti na GitHub pod jednu od OSI odobrenih licenci.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 29. lipnja 2018.

Mentor:
Izv. prof. dr. sc. Mile Šikić

Predsjednik odbora za diplomski rad profila:
Prof. dr. sc. Siniša Srbljić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

I would like to thank my girlfriend Klara Rumora for all the support and help in finishing this work. Moreover, I would like to thank my mentor Mile Šikić for the patience and support he gave me through all the projects we've done together. Many thanks to my friends and family as well, who gave me all the needed support throughout the years.

CONTENTS

1. Introduction

De novo assembly is the process of reconstructing an unknown genome from sequences of DNA acquired by a DNA sequencing output. One of the most modern methods of acquiring such sequences today is shotgun sequencing. This method is the untargeted sequencing of all genomes present in a sample[3][4].

Reconstructing a genome from such single species sample is essentially a task of reconstructing a path from inter-sequence overlaps. In an ideal case, we could reconstruct a genome using only overlap information. However, this is greatly complicated by the presence of repetitive parts. Because these areas are much longer than current sequences acquired by sequencing tools, it is hard to identify them. This can lead to assembly error or fragmentation of the assembly.

The difficulty of the task is even further increased by sequencing errors. These are the result of fast and high throughput of sequencing tools.

In an ideal case, reconstructed genome would have single sequence without any errors or breaks. Unfortunately, most often, this is not the case and reconstructed genome is represented in fragments. These basic elements are called contigs. They are constructed by a set of sequentially overlapping DNA reads.

This task is even further complicated by sequencing a composition of microbial communities[4].

2. De novo Metagenome Assemby

Metagenome assembly poses many additional challenges on top of single genome de novo assembly. It is in many ways conceptually the same as the single genome assembly. Differences between single species assemblers and metagenome assemblers are visible in additional methods of trying to overcome additional challenges. These are caused by sequencing errors, which generate non-genomic sequences, and repetitive sequences. Consequently, this can lead to misassemblies and fragmentation of the assembly[4].

There have been multiple published approaches to reconstruct a microbial community composition from a pool of sequence reads. Selecting the best approach for the job is a difficult task since published results largely depend on the aims of the study[4].

When assembling a single genome, we can usually assume that coverage will be uniform. The assembler can then use coverage data to identify repeat sequences, sequencing errors and etc. Metagenomic assembly is a bigger challenge since coverage of each genome depends on the presence of other genomes in the community. Low-abundance genomes could end up fragmented because overall sequencing depth might not be enough to construct sufficient connections in the graph.

An additional problem is laying in the fact that a sample can contain different strains of the same bacterial species. Such genomically similar species can cause branches in the assembly graph. In such branching, differentiation between two species can be as simple as a single nucleotide variant or absence of some parts of the genome. Bacteria species like NCTC204 and NCTC418 are an example of such problem which we found very difficult to overcome.

Assemblers which are targeted towards metagenome-specific inputs, try to overcome these challenges in multiple different ways. Most of them focus on creating new tools which are tailored for metagenomic input[4].

In this work, we investigate methods of clustering reads from the same species. If done well, this could result in highly precise clusters containing only the reads from the same species. This would make the task of reconstructing a metagenome sequence

similar to the task single genome assembly thus enabling us to reuse existing tools for de novo assembly.

3. Dataset - PacBio/Nanopore

Dataset used in this work consists of several bacteria species samples. They were produces using long-read sequencing technologies such as the Oxford Nanopore MinION and Pacific Biosciences.

To simulate metagenomic input, we joined multiple of these sequences. Most of our test inputs comprised of two combined sequences to investigate how we could separate reads from multiple species when having similar coverages with overlapping areas. We also wanted to observe how well could genomes be reconstructed from the created cluster as there is a possibility of fragmentation when reducing sample size by clustering.

4. Methods

In the following section, we describe methods and tools which were used to implement and test our algorithm.

4.1. Minimap2

Minimap2 is a versatile sequence alignment program that aligns DNA or mRNA sequences against a large reference database. Among others, it is typically used to find overlaps between long reads with error rate up to 15%[2].

We used it in our implementation to generate clustering data for our algorithms such as coverage, number of overlaps and interconnected reads.

4.2. Rala

Rala is a DeNovo genome assembly layout tool. It is intended as a standalone module to assemble raw reads generated by third-generation sequencing.

Additionally, Rala also employs techniques to avoid errors such as chimeric reads which are cut. This is accomplished by scanning coverage graphs and cutting chimeric reads. After preprocessing is complete, an assembly graph is built and simplified with transitive reduction, trimming, bubble popping and heuristics which untangle leftover junctions in graph[6].

4.3. Clustering

We attempted to employ multiple features when clustering reads. However, most widely and simple to use metric were multiple percentile coverages of each read. This was primarily used because of expected similarity in coverages across reads from the same species.

Percentile coverage of the read is percentile value of coverages of all the bases in read. Coverage of a single base is calculated by counting all the overlaps over that base. If median coverage of a read is too low, we instead found the longest covered area in the read which minimum coverages are at least 3. We then calculated percentiles only in that area. This algorithm is similar to the one used in Rala, but we separated it as a standalone module to ease development and to minimize the modifications of Rala module[6].

4.3.1. Naive - K-means

The simplest algorithm to implement with regarding to median coverage of reads was a K-means clustering with median coverages of the read as a single feature.

To determine the number of clusters we ran the algorithm on the size range of $[2, 10]$ and determined the best one by the BIC score.

Since this algorithm performs very poorly when median coverages of multiple species overlap (which they often do, especially in real metagenomic samples), we decided to take additional post-clustering steps to improve our clustering.

The improvement upon simple K-means clustering was done iteratively by moving reads from one cluster to the other. To decide how we transfer read from one cluster to other we used overlaps info from minimap2. The basic idea was to transfer reads to the cluster in which they had most overlaps. This, however, often resulted in large clusters containing reads from multiple genomes. Consequently, the error was most likely in already large mixed clusters, which would sink reads from all the other clusters regardless of the existence of cross-species overlaps.

We tried to maximize the effect by weighing every node in the cluster. The weighing method we used, transferred reads to the cluster with the largest connected graph.

Since this method heavily suffered due to very large multi-species clusters, we tried a different implementation of clustering. The method we decided on is MCL clustering because of its determinism and performance on very large clusters of up to million nodes[5].

4.3.2. MCL

The MCL algorithm is used to find cluster structures in graphs by a mathematical bootstrapping procedure. It simulates random walks through the graph. This is achieved by using stochastic matrices (also known as Markov matrices by which MCL got its

name) that mathematically simulate random walks on a graph. Markov matrix is simply a squared matrix describing probabilities of transitions between nodes. MCL uses Markov chain expansion and additional operator of inflation to emphasize lose and strong bonds between nodes. Expansion corresponds to taking the power of the matrix using the normal matrix product. Inflation corresponds to taking the Hadamard power of a matrix, followed by a scaling step:

$$(M_{pq})^r / \sum_{i=1}^{k} (M_{iq})^r$$

Result matrix of such operation is once again Markov matrix. The inflation is responsible for strengthening and weakening of the bonds between nodes. These two operations are used alternately until convergence at which point, there is no path between some segments. This collection of resulting segments is then interpreted as a clustering. The inflation parameter r controls the extent of strengthening and weakening which in turn influences granularity of clusters[5].

We used existing MCL algorithm implementation. Two mandatory inputs of the MCL tool are graph descriptor file and granularity decimal value. Granularity parameter determines the number of clusters. Selecting the higher granularity parameter will result in larger number of clusters. This is critical to layout step in our pipeline because having smaller clusters will result in lower genome coverage. As a consequence, we run in the risk of fragmenting the genome and even being unable to reconstruct all parts of the genome.

On the flip side, decreasing granularity value results in very large and possible multi-species clusters thus recreating the initial problem we wanted to avoid from the previous implementation.

The parameter of granularity has been chosen empirically in order to find the sweet spot between above-mentioned issues. The purpose was to achieve largest possible clusters without sacrificing too much precision.

Input graph of MCL represented reads as nodes of the graph and overlaps between them as edges. The main challenge of this approach is finding a function which would map reads from the same species closer together as opposed to those from different ones. Granularity parameter was changed depending on the function used for generating input to the MCL.

4.3.3. Overlap classification

Our implementation heavily relies on overlapping data from minimap2 so filtering out bad overlaps (i.e. overlaps between reads from different species), could increase the precision of our clustering methods. To identify inter-species overlaps, we employed classification methods from machine learning and marked bad overlaps as positive matches.

While choosing the appropriate classification algorithm, we looked for the solution that would maximize the recall score. By having the perfect recall score, all of the inter-species overlaps would be removed. However, impaired precision would degrade the performance of the layout stage. Consequently, we decided to set the hard limit on the lowest possible precision to 50% to avoid such degradation in final results.

We improved our pipeline by removing all positively classified overlaps from overlaps input file which is used in clustering algorithms and Rala. Even though filtered out overlaps can be used again in determining the coverage of the read, we didn't employ this strategy as it would likely bring no major improvement and possibly even deteriorate the performance of the pipeline.

5. Assessing results

To assess our algorithm, we joined sets of reads from multiple different single-genome bacteria sources. Input was first run through a clustering algorithm and each resulting cluster was afterward processed through Rala.

Results assessment is done in two stages. First one is performed immediately after clustering and is used to assess the precision of each cluster in accordance to the most representative reference. This is done by counting the most representative reads in that cluster and dividing it by the total amount of reads.

However, because reads of a single reference can span across multiple clusters, it is important to check whether this sample size is qualitative enough to construct the genome. To achieve this, we ran each cluster through Rala and mapped all contigs back to original references. Only highly mapped references (maximally 10% less coverage than Rala ran on the original input for that reference) are then taken into consideration for the observance. Observed features on results are mean contig purity, mean contig purity of mixed contigs, lowest contig purity, contaminated length of contigs, number of contigs, count of mixed contigs and NG50 rating and purity of NG50 and NG90 contigs. Contig purity p is defined similarly as cluster accuracy. The maximum length covered by a single source reads is divided by the total contig length l. Contaminated length of a contig is calculated as follows:

$$(1 - p) \cdot l$$

Contig purity and total contaminated length of contigs are useful to inspect how much improvement, if any, is achieved while using clustered metagenomic input. For example, contig purity can be a good indicator of overall cleaner contigs, while the lower total contaminated length is a definitive indicator of better contig quality across the whole cluster.

We also inspected purity of contigs used in NG50 and NG90 scores as those are good indicators of how clean would be the reconstructed genome if there wasn't any fragmentation.

To measure effects of the clustering algorithms, we compared these results to two different baselines. First one is the result of Rala run with combined input and no clustering. We can observe same features with this method. The second one is the result of Rala run separately for each joined input. This way, we can observe our results in comparison to the imaginary outcome of a perfect clustering

6. Implementation

All developed algorithms were implemented in Python3 and can be found at GitHub with exception of upgrades to Rala, which is written in C++ and is included as a submodule.

To increase speed and boost ease of development, implementation is divided into many simpler scripts. These scripts are then combined together in a pipeline (Figure 6.1) that performs all the necessary steps from joining the reads as a simulated input to layout phase with included metric results.

We used two different clustering methods. First one is a k-means clustering approach by grouping reads around same percentile (e.g. median) coverage and additional adjustment. The second one used MCL algorithm and input utilizing read overlaps, percentile coverage of reads and other overlap information.

6.1. Naive clustering implementation

Naive clustering implementation used K-Means implementation from scikit-learn library.

Ideally, all reads from the same species should have the same coverage, so this approach could group reads together very well if the coverages of the species in the metagenomic sample differentiate enough. However, since coverages of a single species often span across multiple coverage values rather than spiking in a single one, it is commonly observed that multiple species share some coverage span (Figure 6.2).

It is also hard to disregard high error rates of sequencing methods and natural occurring of false overlaps between unconnected segments of a genome. Due to these effects, we implemented cluster correction step in which we iteratively move reads between clusters. Correction algorithm that proved to be the best in this scenario used information about read overlaps and interconnection of the reads within a cluster. For each read in a cluster, we calculated the total number of the reads in a connected graph in that cluster. This value would represent the weight w of each node in the cluster.

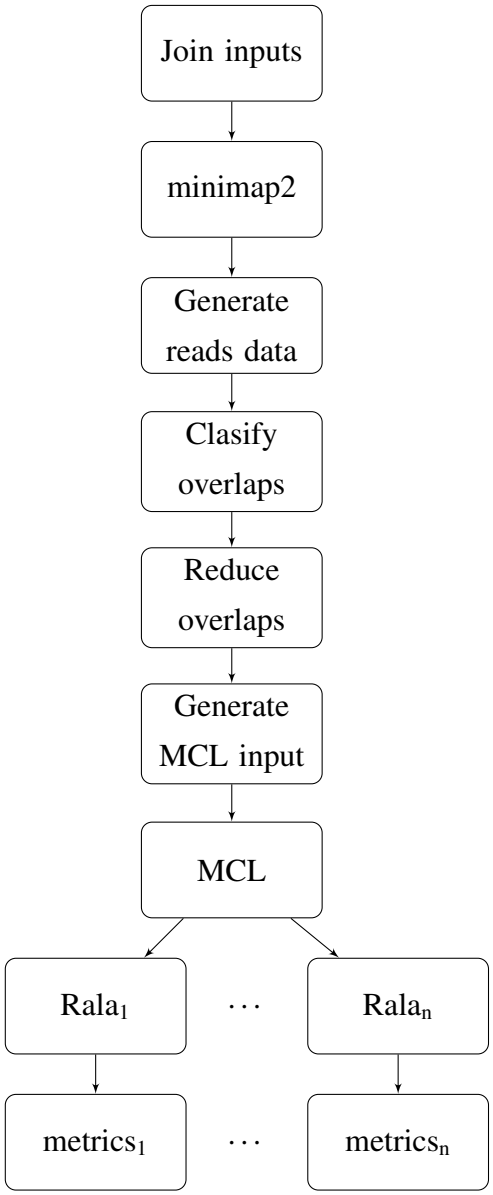

Figure 6.1: Flow chart of the pipeline of our complete solution.

The correction step would move any read from i cluster to j cluster based on the total sum of the weighted connections. The goal is to find the cluster with maximum $p[i, j]$ which is the cost of having i-th read in the j-th cluster. This cost is calculated by summing up all the weights of the nodes to which the read i is connected in each cluster. Weight is represented as the total amount of nodes in the graph in which particular node belongs. Example is given in Figure 6.3.

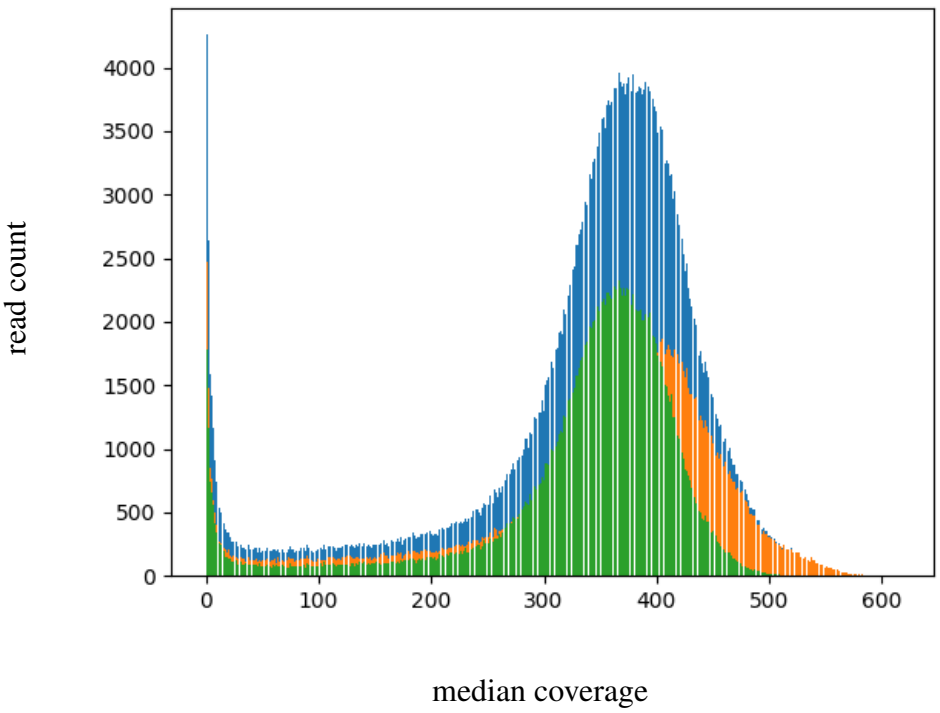

Figure 6.2: Example of overlapping median coverages histograms for different genomes. In such situations, it is impossible to separate reads from different genomes just by coverage information. Green and orange histograms represent median histogram coverages of two different genomes, while the blue one is their aggregate.

6.2. MCL clustering implementation

Our more advanced clustering implementation uses MCL. In this implementation, reads were represented as nodes and their overlaps as edges. The idea behind this approach was to find a way to separate reads in highly connected and dense clusters. Because this is the core idea of MCL, we tried to find a function which would map reads from the same species closer together as opposed to those from different ones.

We initially used granularity of 1.09 which was chosen empirically. However, it may be needed to adjust this value for larger input sets.

Values between edges represent only varying formula in this approach. We tried few different methods to separate different species as much away from each other.

To build upon the idea of naive clustering, we used a similar approach in first MCL implementation. We calculated edges as an absolute difference between overlapping percentile coverages. This approach appeared to be better in many cases than naive

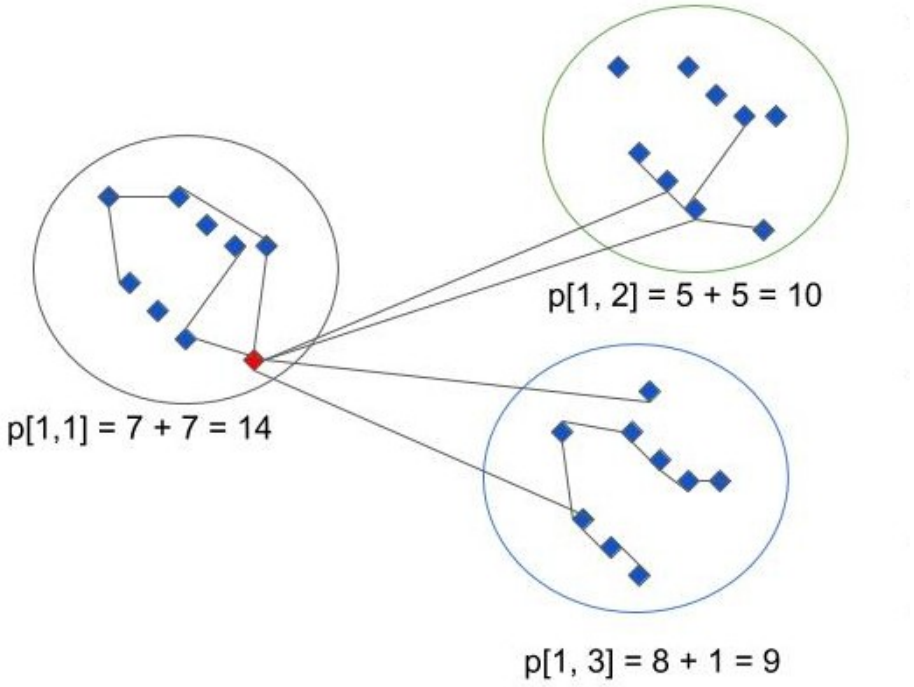

Figure 6.3: Example of naive clustering iterative correction step. Cost of having red node in the existing cluster is the highest of costs across all clusters and doesn't require transfer.

approach and it was also much more consistent.

However, since this approach also didn't have greatly improved results over results of Rala ran without clustering, we sought other metrics to define values of the edges between reads in MCL clustering.

These additional approaches were greatly inspired by classification step in which we explored various features that can be used in detecting invalid overlaps.

Since MCL doesn't have a way to deal with outliers, there is a significant amount of sparse (small) clusters. We ignored any clusters which contained less than 1000 reads. These reads could be reassigned to larger clusters by using similar methods of transferring reads in our naive implementation. We haven't attempted this, however, because the amount of reads in such clusters is significantly small (<1% of total sample) therefore it most likely would not have a great impact on the results of the pipeline.

6.3. Classifying overlaps

We tried three different classification algorithms – Naive Bayes, SVM with RBF kernel and Random Forrest.

Random Forrest proved to be the best (Figure 6.4) for this problem since it produces similar or slightly worse recall scores as the other two but significantly higher precision scores. It is worth noting that although we tried to optimize for highest recall score, precision score still plays an important part in the outcome. If we remove too much fo the correct overlaps, we're risking the fragmentation of the assembly. In extreme situations, this could even lead to the absence of some parts of the genome because there aren't enough overlaps to reconstruct the path. Because of these issues, we opted for the Random Forrest because it offered acceptable precision while maintaining decent recall scores.

Features of the final implementation used the minimum and the maximum difference of two percentiles of read coverage, the difference of overlap count on the overlapping side, percentile coverages difference and overlap quality. We defined the quality of the overlap as a ratio between the number of residue matches and the alignment block length of minimap2 output[1].

This implementation wasn't thoroughly investigated and requires more research as it wasn't in the focus of our study.

Classification algoirthm	precision	recall
Random Forrest	0.392	0.020
SVM	0.271	0.141
Naive Bayes	0.303	0.003

Classification algoirthm	precision	recall
Random Forrest	0.410	0.092
SVM	0.386	0.363
Naive Bayes	0.394	0.023

Classification algoirthm	precision	recall
Random Forrest	0.733	0.237
SVM	0.192	0.660
Naive Bayes	1.0	0.000

Classification algoirthm	precision	recall
Random Forrest	0.767	0.259
SVM	0.140	0.502
Naive Bayes	0.825	0.000

Figure 6.4: Classification stage tested on 4 different samples with Random Forrest, SVM and Naive Bayes. Tests were done independently by splitting each sample in train and test sets in $1 : 99$ ratio.

7. Results

7.1. Clustering results

To assert that our advanced implementation is truly better than the naive implementation, it is required that the total error rate of all clusters is measured. This is calculated by summing up all of the reads which are a minority in each cluster and dividing it by the total amount of reads. We compared our initial MCL implementation to the naive one and observed significant improvements (Table 7.1). Initial implementation had the granularity parameter set to 1.10 and used the difference of median coverages as edge value.

Table 7.1: Cluster precision results using naive and MCL methods.

Joined bacteria	MCL method	Naive Method
NCTC74, NCTC235	1.32	19.43
NCTC74, NCTC2218	0.04	6.03
NCTC74, NCTC3750	0.06	6.31
NCTC204, NCTC235	18.80	11.28
NCTC204, NCTC2218	0.03	6.50
NCTC204, NCTC3750	0.02	5.43
NCTC235, NCTC418	31.89	34.55
NCTC235, NCTC1080	0.05	5.27
NCTC418, NCTC2218	0.07	6.45
NCTC418, NCTC3750	0.08	6.53
NCTC1080, NCTC2218	0.07	39.52
NCTC1080, NCTC3750	0.06	17.81

It is proven to be more precise in most examples. Moreover, it is more consistent as opposed to our naive approach in which results can vary depending on the run. This is

due to the indeterministic trait of the first stage of clustering. The second stage of naive clustering heavily depends on the first one and tends to transfer most of the reads into a single large mixed cluster. This often occurs when the input has a large overlapping area of coverages (6.2).

However, we didn't track the count of clusters because each implementation produced an acceptable number of clusters in the range of $[2, 10]$. It is also possible that purer examples like joined reads from NCTC204 and NCTC2218 sequences could be further improved by decreasing the granularity parameter.

However, this was not the target of our study as these examples didn't have any mixed contigs even in the baseline results. In one such example, we joined 4 reads together which showed to be easily mutually separable and still got no mixed contigs from the output of layout step.

7.2. Comparison of MCL edge values

Our initial implementation utilizing MCL algorithm valued edges of the graph as the difference of the percentile coverages of the reads. This worked excellent for the sequences which have very high average difference of percentile coverages (Table 7.1, Figure 7.1).

To improve our solution, we also tried removing overlaps from the MCL input which are not strictly overlapping on one side. These overlaps are also filtered in Rala, but removing these overlaps earlier would erase illegal bonds in the graph and could potentially create cleaner clusters. We observed that, in some of our examples, roughly half of the illegal overlaps can be removed with this approach. However, this also removed just about the same ratio of the legal overlaps. The following formula was used for determining overlaps on strictly one side:

$$f(x) = \begin{cases} 1, & \text{if } query_{left} > target_{left} \wedge query_{right} < target_{right} \\ 0, & \text{if } query_{left} > target_{left} \wedge query_{right} < target_{right} \\ -1, & \text{otherwise} \end{cases}$$

where $(query/target)_{left/right}$ defines trailing leftover before or after the overlap on query or target and positive result values represent strict overlapping on one side (1 for right overlap on query and 0 for left overlap on query read).

This approach doesn't yield significantly different results. Comparing clustering purity gives almost the same cluster count and distribution (Figure 7.2). More interest-

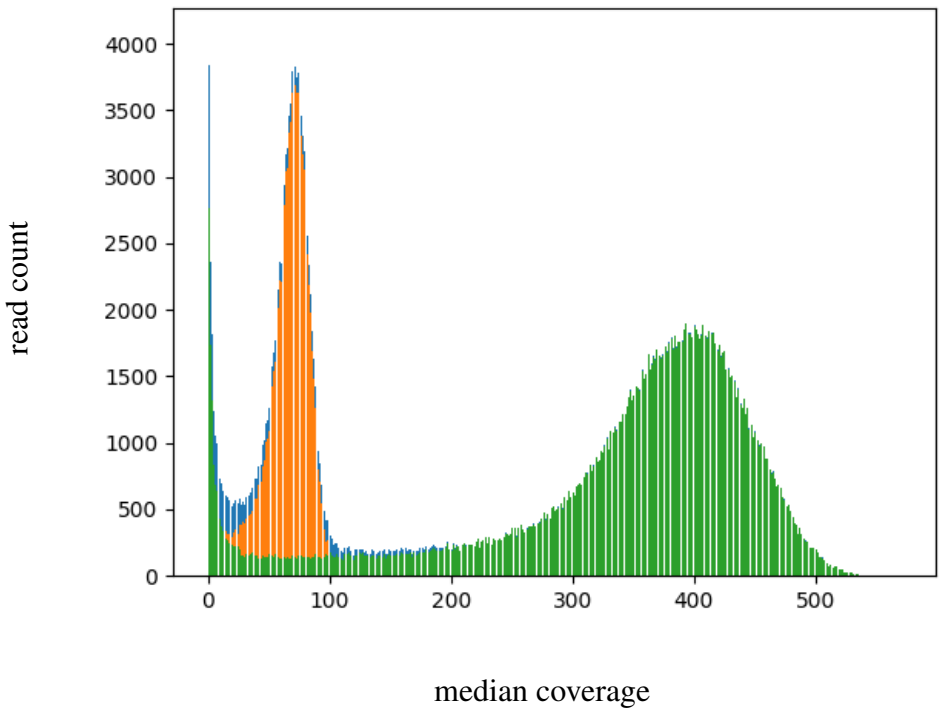

Figure 7.1: Example of two genomes easily separable on the basis of their median coverages. Orange and green histograms represent median histogram coverages of NCTC 235 and NCTC 1080 bacteria respectively, while blue one is their aggregate.

ingly, the first implementation of MCL (using all edges) yields better NG50 and NG90 coverages in most cases (Table 7.2).

There are two issues with the approach of using the difference between percentile coverages as edge values. Firstly, there is no clear indicator that overlapping reads of the same species should have similar coverages in metagenome assembly. We could expect this in a single species sample without errors. However, in a multispecies sample, this is disrupted by error in sequencing, repetitive sequences and false overlapping by a multi-species sample (Figure 7.3). Secondly, having overlapping regions of same percentile coverage values could result in bringing inter-species reads closer together in MCL input graph. To avoid this, we introduced other metrics. First one is the maximum difference between two percentile coverages. For example, we took maximum difference of 50^{th} and 90^{th} percentile coverage.

$$max(abs(a\_cov_{90} - b\_cov_{50}), abs(b\_cov_{90} - a\_cov_{50}))$$

The second added feature was the difference in overlap count on the overlapping side.

Table 7.2: Sample comparison of layout results using base method and base method using strictly overlapping edges.

Joined input set	Reference genome	Rating	1st method	2nd method
NCTC204, NCTC235	NCTC235	NG50	1	4
		NG90	1	5
	NCTC204	NG50	7	13
		NG90	15	31
NCTC235, NCTC418	NCTC235	NG50	2	5
		NG90	4	5
	NCTC418	NG50	5	1
		NG90	N/A	N/A

To combine these values together as a single value in the edge, we took the median relative value of these three features as shown in Figure 7.5. We also tried approaches with minimum and maximum values of said three features, but median has shown to be the best choice in our samples.

This approach resulted in a significantly smaller number of clusters while maintaining approximately same precision score across clusters (Figure 7.4). In this approach, granularity value of 1.12 was used. In order to compare the quality of clusters generated by both methods, we tried matching the number of clusters in base MCL approach and the new one. We attempted this by trying to reduce the number of clusters in base approach. However, cluster count didn't change significantly after granularity value was decreased by 0.02.

The contaminated length was smaller and median purity of NG50 and NG90 contigs were overally higher in most cases. However, this approach resulted in higher fragmentation represented by deteriorated NG ratings in nearly all cases despite having larger clusters.

We also tried multiplying each edge value with the squared quality of the overlap. This increased number of clusters once again and deteriorated overall precision of clusters. Decreasing granularity parameter decreased the number of clusters once again, but retained worse cluster precisions.

Our best approach maintains same species coverages as Rala ran on single input sequences. It also somewhat improves on the Rala ran without clustering. In most samples, we observed much cleaner contigs (Table 7.3). However, additional fragmen-

Cluster precision	1st species read count in cluster	2nd species read count in cluster
0.6267	26405	44335
0.7340	10425	28768
0.7232	7158	18706
0.5870	6789	9651
0.5842	4982	6999
0.7013	2830	6643
0.7003	2770	6474
0.9999	8967	1
0.5087	3898	3764
0.7477	1604	4754
0.8739	573	3970
0.5597	1306	1660
0.6121	713	1125

(a) removed non-strictly overlapping edges

Cluster precision	1st species read count in cluster	2nd species read count in cluster
0.6307	23828	40696
0.6584	13008	25075
0.7187	10591	27064
0.5869	7162	10177
0.6926	5158	11619
0.9998	9514	2
0.6925	2591	5834
0.5448	4528	3784
0.7593	1447	4564
0.6768	1892	3962
0.9997	2	5841
0.5640	1314	1700

(b) base MCL edge values

Figure 7.2: Comparison of MCL edge values clustering results

tation occurred in our approach as expected.

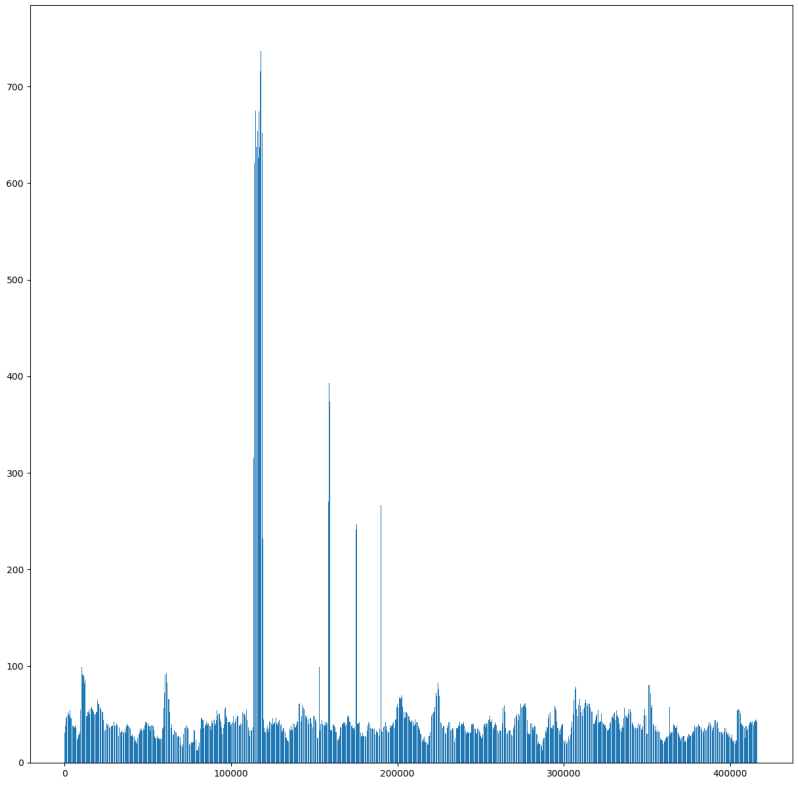

Figure 7.3: Coverage graph of a sample contig produced by Rala.

Cluster precision	1st species read count in cluster	2nd species read count in cluster
0.9648	1486	40778
0.9989	24789	27
0.9742	426	16071
0.7785	3342	11746
0.9017	1448	13279
0.9412	654	10468
0.9996	4	10371
0.9277	596	7642
0.9477	405	7345
0.9989	7314	8
0.8005	1045	4193
0.9998	1	5162
0.952	207	4104
0.999	3	2885
0.9995	2063	1

(a) base MCL edge values

Cluster precision	1st species read count in cluster	2nd species read count in cluster
0.9708	2196	72948
0.8454	11195	61240
0.9993	14677	10
0.9995	12789	6
0.999	2929	3

(b) median of three features edge values

Figure 7.4: Comparison of MCL edge values clustering results.

```
# pcov_x - x percentile coverage of the read
np.median([
  abs(
    info[a_id].pcov_10 - info[b_id].pcov_10
  ) / max(
    info[a_id].pcov_10, info[b_id].pcov_10, 1
  ),
  max(
    abs(
      info[a_id].pcov_90 - info[b_id].pcov_50
    ) / max(
      info[a_id].pcov_90, info[b_id].pcov_50, 1
    ),
    abs(
      info[b_id].pcov_90 - info[a_id].pcov_50
    ) / max(
      info[b_id.]pcov_90, info[a_id].pcov_50, 1
    )
  )] + ([
  abs(
    (
      info[a_id].rightside_overlaps -
          info[b_id].leftside_overlaps
    ) / max(
      info[a_id].rightside_overlaps),
          info[b_id].leftside_overlaps), 1
    ) if side == 0 else \
    (
      info[a_id].leftside_overlaps -
          info[b_id].rightside_overlaps
    ) / max(
      info[a_id].leftside_overlaps,
          info[b_id].rightside_overlaps, 1
    )
  )] if side != -1 else [])
)
```

Figure 7.5: Algorithm for calculating edge values for MCL in our final approach

Table 7.3: Sample comparison of layout results using MCL clustering method and baseline method of running Rala without clustering.

Joined input set	Reference genome	Rating	Our approach	Baseline
NCTC235, NCTC418	NCTC235	NG50	4	4
		NG90	4	4
	NCTC418	NG50	2	2
		NG90	N/A	N/A
	Maximum total contaminated length		317848	1954764
NCTC74, NCTC204	NCTC74	NG50	5	3
		NG90	9	N/A
	NCTC204	NG50	17	9
		NG90	N/A	N/A
	Maximum total contaminated length		68114	2085259

8. Discussion

Naive cluster implementation heavily suffered due to very high possibility and often occurrence of creating large clusters of multiple species. In such situations, the first stage already resulted in highly mixed and large clusters. This would be even further emphasized in the second stage where most of the reads would be moved to the exact same large cluster, creating a super-large multi-species cluster containing most of the sample reads.

MCL has easily proven to be better at clustering reads in this situation. However, its weakness lays in a large number of illegal overlaps which are translated as edges in the MCL input. This issue could be resolved by removing additional edges. It is worth noting that some of the joined sequences have some amount of false overlapping, which suggests that perfect classification of illegal overlaps may not be needed to achieve highly accurate clustering results. Second, and maybe significantly easier way to improve the clustering would be exploring additional mapping functions for the values between edges. MCL does a great job in separating clusters deterministically, but values between edges should be chosen carefully. If done wrong, MCL clustering can result in large multi-species clusters similarly to naive implementation.

One of the downsides of all our approaches to clustering with MCL is that species with very high overlap in the genome will be easily grouped together just on the basis of connectivity between clusters. In these situations, it is impossible to separate clusters based on the features we used. This is best represented in case of joined input of NCTC204 and NCTC418 which are genomically similar species and have a large number of interspecies overlaps.

Selecting higher granularity values would improve the cluster precision as well as contig purity but would result in more fragmented contigs. It would also result in a fragmented clustering, meaning that we couldn't have reconstructed a genome from a single cluster but would rather need to group some clusters back together. This leads to other problems since it would be hard to identify clusters which contain the same species contigs. One possible solution to this problem is to identify clusters which

would be mapped to known genomes with high accuracy. However, this requires great time and space complexity of the algorithm.

9. Conclusion

Metagenomic sequencing poses many challenges. Reads clustering may be a sufficient way to solve this issue. Our final implementation improved the results of Rala tool by creating cleaner contigs. The tradeoff of this approach is additional fragmentation of the assembly.

However, it cannot reliably separate clusters that have similar genomic sequences. Moreover, it is difficult to choose cluster count. By generating larger cluster count than required, the assembly is additionally fragmented. This is a recurring problem in all our approaches.

Both, the MCL input and classification of invalid overlaps, could be further improved. Classification of invalid overlaps wasn't thoroughly researched and could give a significant performance boost to our approach. MCL could be improved as well, by further investigation of the mapping functions for the computation of the edges.

BIBLIOGRAPHY

[1] Heng Li. miniasm. URL https://github.com/lh3/miniasm/blob/master/PAF.md.

[2] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, stranica bty191, 2018. doi: 10.1093/bioinformatics/bty191. URL http://dx.doi.org/10.1093/bioinformatics/bty191.

[3] Mirjana Domazet-Lošo Mile Šikić. Bioinformatika - skripta. 12 2013.

[4] Christopher Quince, Alan W Walker, Jared T Simpson, Nicholas J Loman, i Nicola Segata. Shotgun metagenomics, from sampling to analysis. 35:833–844, 09 2017.

[5] Stijn van Dongen. Graph clustering by flow simulation. 2000.

[6] Robert Vaser. *Rala*. Faculty of Electrical Engineering and Computing, 2018. URL https://github.com/rvaser/rala.

De Novo Metagenome Assembly Using Read Clustering

Abstract

Metagenomic assembly poses many additional challenges over single genome assembly. In this work, we tried tackling these issues by clustering reads produced by third-generation sequencing tools. In an ideal case, this would allow the use of existing de novo assembly tools for single genome assembly on metagenomic input. Our clustering implementation showed some improvement of existing tools for de novo assembly when using them for metagenome assembly, but still requires a lot of research. The source code of the solution is available at https://github.com/lukadante/rala-cluster.

Keywords: metagenome assembly,clustering,DNA

De novo sastavljanje metagenoma koristeći metode za grupiranje očitanja

Sažetak

Kod metagenomskog sastavljanja, suočeni smo s mnogim dodatnim izazovima u odnosu na sastavljanje jednog genoma. U ovom radu, pokušali smo riješiti ove probleme grupiranjem segmenata koje proizvode alati treće generacije za sekvenciranje. U idealnom slučaju, to bi omogućilo korištenje postojećih de novo alata za sastavljanje genoma na metagenomskom uzorku. Naša implementacija grupiranja je donekle poboljšala postojeće alate za de novo sastavljanje kod primjene na metagenomskom uzorku, no još uvijek su potrebna daljnja istraživanja. Izvorni kod rješenja dostupan je na adresi https://github.com/lukadante/rala-cluster.

Ključne riječi: metagenomsko sastavljanje,grupiranje,DNA