

Graph pre-processing strategies for minimum flow decomposition in haplotype aware genome assembly

Senne Drent

Supervisor: Jasmijn Baaijens

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
January 25, 2026

Name of the student: Senne Drent
Final project course: CSE3000 Research Project
Thesis committee: Jasmijn Baaijens, Petr Kellnhofer

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Minimum Flow Decomposition (MFD) can be used as a tool for assembling haplotypes in a viral sample. This application of MFD uses a contig variation graph as input, created from DNA sequencing reads. MFD is NP-hard and runtime grows exponentially with increasing haplotypes, making runtime reduction crucial. Noise in the input graph results in slower runtimes and decreased accuracy. This research investigates whether the three different pre-processing strategies in the contig variation graph reduce runtime and increase accuracy.

First, applying abundance correction to mitigate non-uniform coverage of the sample. This strategy shows a 15.8% faster runtime on a 6-strain dataset, without affecting accuracy.

Second, evaluating different configurations for the low-abundance node filter thresholds. An optimal threshold is found which does not impact accuracy but reduces runtime by 25%.

Third, the proposed cycle-resolution algorithms are not suitable for breaking the assembly error cycles in the contig variation graph.

Overall, results show that targeted preprocessing of contig variation graphs can improve MFD performance, while highlighting the need for cycle-resolution methods tailored to haplotype assembly graphs.

1 Introduction

In bioinformatics, genome assembly is the process of combining many pieces of raw DNA data to reconstruct genomes. Genomes contain genetic information of an organism or cell. In the context of viruses, a single sample may consist of distinct copies of the genome, also referred to as haplotypes. This is through relatively quick mutation of viruses.

Different haplotypes can have different properties, such as resistance to medicine. Making reconstructing viral genomes essential for further understanding of viruses and improving treatment options.

Viral samples are read by DNA sequencers. Sequencers do not read DNA samples in one pass, but perform short reads for increased accuracy. Meaning these reads need to be pieced together without knowing what data belongs to which haplotype.

This read data can be encoded into a contig variation graph, nodes with abundance describe subsequences from the DNA reads and edges represent how the subsequences can be combined. Contigs are longer DNA sequences assembled from overlapping short reads, serving as subpaths which should be included in haplotypes in its entirety.

Assembling haplotypes can be modeled as a Minimum flow decomposition (MFD) problem. MFD is defined as decomposing a graph into a set of weighted paths, such that for every edge e, the sum of weights from the paths along e equals the weight of e. The contig variation graph is used as input, and each output path from the MFD represents the genome of a haplotype and the weight of the path its frequency in the sample.

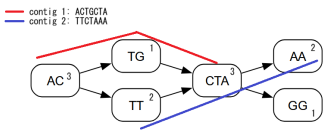

Figure 1: Example of a contig variation graph

The MFD problem is known to be NP-hard [9]. For genome assembly, runtime grows exponentially with the amount of haplotypes to assemble [11]. Though, recent improvements [6] show this approach competes with the currently available tools in terms of assembly quality [11]. However, since virus samples can contain hundreds of haplotypes it is crucial to look into further optimizations and runtime reduction, while maintaining accuracy of the assembly.

Errors are introduced into the contig variation graph due to noise in DNA reads and steps in the pipeline. One such error is the violation of flow conservation in a graph. Worse violations could lead to slower runtimes and reduced accuracy. Another type of error in the graph are nodes with little abundance, these are considered to be noise. These nodes are filtered by setting a threshold defining when the abundance is too low. It is not clear however at what point a node should be considered erroneous. Finally, this approach needs a DAG as input for the solver. In the current pipeline however, cycles can be introduced into the contig variation graph. Caused by alignment errors of DNA sequences or repeats. The current pipeline is not built to resolve these cycles automatically, cycles are removed by visual inspection and manual removal.

This research attempts to answer the following main question: how can a contig variation graph be pre-processed to improve performance of the Minimum Flow Decomposition for haplotype assembly?

To answer this, the following subquestions are researched:

- How can noise in the abundance be corrected using the coverage of DNA reads at certain positions in the viral sample?
- How do different thresholds for low-abundance noise filtering impact the solution in runtime and accuracy?
- How can cycles be resolved efficiently without impacting the solution results?
- What is the effect of automatic cycle resolution on the accuracy of the MFD?

To answer these, three different strategies are implemented and analyzed by impact on the graph and solution. One strategy proposed is an algorithm to correct abundances based on DNA read coverage bias. Another strategy is low-abundance node filtering, different thresholds will be evaluated to find an optimal configuration with respect to runtime and accuracy. Finally, the research proposes two different algorithms for cycle resolution and evaluates their ability to break cycles in a contig variation graph.

2 Background

MFD is NP-hard, runtime grows exponentially as the number of haplotypes increases [11]. Furthermore, MFD can only converge to a solution when the graph satisfies flow conservation. Due to assembly errors and read noise, flow is not always conserved in a contig variation graph.

To get a solution from graphs without the flow conservation property, k least absolute error (KLAE) is used. Given k, KLAE finds a solution set of paths $P_1, ..., P_k$ each with a weight w_i, minimizing the following objective function:

$$\sum_{(u,v)\in E} \left| f(u,v) - \sum_{i\in\{1,...,k\}:(u,v)\in P_i} w_i \right| \quad (1)$$

Contigs are modeled as subpath constraints, meaning the subpath has to be fully contained in at least one of the solution paths.

To choose k, Minimum path cover (MPC) is used to estimate the number of haplotypes in a contig variation graph. MPC is defined as the minimum amount of paths needed to cover the whole graph.

2.1 DNA read coverage

One way of gathering DNA data is by performing Next Generation Sequencing (NGS). This method performs multiple smaller reads, from these reads the contig variation graph is built. DNA read sequences have overlaps, the amount of overlaps in a region of the viral sample indicates how much that region has been read, this is called coverage. Ideally all the areas are read uniformly and thus have similar coverage. This however is not always the case, which can be seen in Figure 2. Thus if abundance is correlated to coverage, then abundance can be underestimated by a lack of coverage in DNA read data. This is a type of error I aim to correct because incorrect abundance causes violation of flow conservation and potential error in the frequency of the assembled haplotypes.

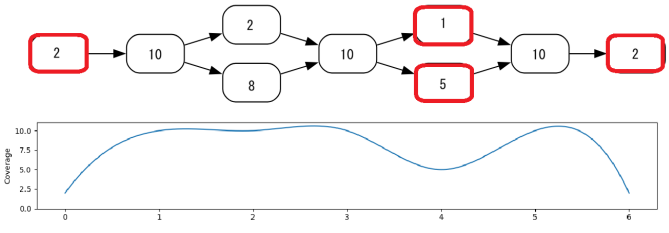

Figure 2: Illustration of coverage bias, values inside the nodes represent their abundance. The nodes marked red violate the flow conservation. The coverage plot represents coverage at each region of the graph directly above.

2.2 Cycle resolution

Cycles in graphs are created from assembly errors and repeats in the DNA sequences, examples of this have been found in [11]. Resolving these cycles typically involves two steps. The first step is inferring a hierarchy in the graph, approximating a topological order as an exact ordering cannot be created with cycles. The second step is cutting edges based on heuristics defining what a violation in the hierarchy is.

Minimum Feedback Arc Set problem

The minimum feedback arc set (MFAS) problem is defined as finding the minimum number of edges to make the graph acyclic. If cycle-resolution in a contig variation graph can be modeled as a MFAS problem, algorithms solving MFAS are applicable. It can only be heuristically solved, as this problem is known to be NP-complete [12].

H-voting

H-voting [14] is an algorithm that breaks noisy hierarchies on an ensemble of strategies. It uses heuristics to solve the MFAS problem described above.

Hierarchy is inferred using two methods. First, TrueSkill [10] treats nodes as players and edges as matches, updating rankings based on outcomes. Second, social agony [7] assigns rankings to minimize "agony", reflecting the tendency of higher-ranked nodes to receive edges from lower-ranked ones.

Then Strongly Connected Components (SCC) are found, for each SCC three different cutting strategies are proposed: greedy, forward and backward. Greedy chooses the edge with the most difference in hierarchy score. The forward strategy takes the highest ranked node from the SCC, and removes all the outgoing edges. Finally, the backward strategy takes the lowest ranked node and removes all incoming edges.

With 2 hierarchy inferring algorithms and 3 strategies deciding cuts, each SCC will be cut according to a voting scheme of the 6 outputs.

3 Methodology

Building on the existing literature, I will explore three different methods of pre-processing, each addressing different aspects or problems. Implementing these with the goal of improving the runtime and accuracy for KLAE.

3.1 Pipeline adjustments

I first used the pipeline described in [11] on the SAVAGE benchmark datasets to create the contig variation graph. However, KLAE did not solve in 24 hours and did not have a feasible solution when k was equal to the number of haplotypes in a strain.

The cause of this was that the contig filter threshold was set too low. This stems from contigs abundances, estimated using Kallisto [3], not having the same scale across datasets. However, filtering was done with a predefined number, missing nodes just above this but still many times smaller than the average contig.

To solve this, I implemented a contig filter with a threshold based on a percentage of the maximum contig abundance. This way a scale is applied to when contigs are considered to be erroneous.

3.2 Abundance correction

To remedy error based on coverage bias, I came up with a method to rescale the abundances according to coverage. An overview of the steps is described in algorithm 1. First a path from the contig variation graph is generated, which serves as a reference of coverage for regions of the graph. Then an alignment is created by running BWA-mem2 [15] using the

generated reference path and the DNA short reads as input. This alignment is parsed to get the coverage for each base pair in the reference path. The coverage of this reference path is then used to map coverage to the rest of the nodes in the graph. Finally, using the coverage metric node abundances are scaled to correct errors originating from coverage differences.

Algorithm 1 Correct abundance in a Contig Variation Graph

function CORRECTABUNDANCES(G)
  $\mathcal{P} \leftarrow GenerateReferencePath(G, strategy)$
  Initialize empty string $\mathcal{S}$
  for each node $v_i \in \mathcal{P}$ **do**
    Append the base-pair sequence represented by v_i to $\mathcal{S}$
  end for
  Save $\mathcal{S}$ as a FASTA file $\mathcal{F}$
  Run `BWA-mem2` alignment on $\mathcal{F}$ with DNA read data
  $\mathcal{C} \leftarrow (c_1, c_2, \ldots, c_{|\mathcal{S}|})$ from alignment results ▷ c_i is the coverage at position i in $\mathcal{S}$
  $G' \leftarrow$ BWA-MEM2-MAPPING$(G, \mathcal{C}, \mathcal{P}, \mathcal{S})$
  for each node $v \in G'$ **do**
    SCALEABUNDANCE$(G', v, \mathcal{C})$
  end for
  return G'
end function

Choosing a reference path

I implemented two strategies to generate a reference path which is used to get coverage information. The first strategy is a path created by a greedy heaviest path, always choosing the largest abundance. This was chosen as it is quick and gives a path with a large abundance. Thus the path should be a decent representation of the viral sample as a whole.

The second strategy is choosing the path which contains the longest base-pair sequence in the graph. Since the graph is a DAG, this can be done efficiently by using topological sort. This was chosen as the longest DNA string in a graph is likely close to the size of most of the haplotypes, representing the coverage of the sample well.

Mapping the coverage to the contig variation graph

Coverage is mapped to the graph by aligning each node to the reference path using BWA-mem2 as described in Algorithm 2. To get these alignments, a path is taken on which the node-to-align lies in the center. This path is constructed by taking a walk along its incoming edges ("backwards") and outgoing edges ("forwards"). Nodes on the reference path are given priority in this walk as paths containing sequences identical to the reference provide stronger alignment anchors and therefore produce better alignments. After this the base-pair sequences from the created paths are put into a FASTQ file. Then BWA-mem2[15] is used to align the file data to the reference sequence. This alignment indexes each node and places it along the reference sequence. Finally, this indexing is used to get the coverage for every node in the graph. This coverage is obtained by looking at the coverage data for the reference sequence at the region the node is indexed at.

Algorithm 2 Map Coverage to Graph Using BWA Alignment

Input Graph $G = (V, E)$, Reference path $\mathcal{P} \subseteq V$, Reference genome sequence `ref_seq`
Output Graph G with `refIndex` for each node
function BWA-MEM2-MAPPING(G, $\mathcal{P}$, ref_seq)
  $\ell \leftarrow 0$
  for all nodes $v \in \mathcal{P}$ in order of path **do**
    $G[v]$.`referenceIndex` $\leftarrow \ell$
    $\ell \leftarrow \ell + |G[v]$.`sequence`$|$
  end for
  `diverging_nodes` $\leftarrow V \setminus \mathcal{P}$
  `paths_to_align` $\leftarrow \emptyset$
  for d in `diverging_nodes` **do**
    Create path $R \leftarrow \{d\}$
    From d, repeatedly prepend a predecessor to R (preferably from $\mathcal{P}$)
    From d, repeatedly append a successor to R (preferably from $\mathcal{P}$)
    Add R to `paths_to_align`
  end for
  Convert $paths_to_align$ to FASTQ reads
  Run BWA-MEM2 aligner on $paths_to_align$ and ref_seq, returns $alignment_file$
  for all nodes $v_i \in \mathcal{P}$ **do**
    Parse `alignedIndex` for v_i in $alignment_file$
    $G[v_i]$.`refIndex` $\leftarrow$ `alignedIndex`
  end for
  return G
end function

Choosing scaling function

The following scaling function is used to scale the abundance for every node n:

$$A'_n = A_n * (\mu_c / c_n) \tag{2}$$

- A is the original abundance of node n
- μ_c is the average coverage of the reference sequence
- c_n is the coverage of the node n
- A'_n is the rescaled abundance for node n

Using this node abundance will be scaled proportional to how much the coverage deviates from the average.

3.3 Cycle resolution

For cycle resolution 2 algorithms are evaluated by keeping track of how many edges are correctly removed and by how many incorrect edges are removed. An edge is correctly removed when the edge removed created a backward connection. The backward edges have been logged by manual inspection of the graph.

Depth first search resolver

One strategy to resolve cycles using a Depth first search (DFS) to search through the graph for cycles. Hierarchy is recursively defined as follows: if the node is not visited, it will get the hierarchy of its predecessor + 1. When the current node is already visited it will check if:

- Its predecessor has a higher or equal hierarchy
- There is a path from the current node to the predecessor

Once these conditions hold, then there is a cycle detected and it should be removed. It is not guaranteed however it removes a cycle once cutting an edge. This method was mainly chosen for its simplicity.

H-voting

This algorithm was chosen as the paper mentions that it is applicable for general purposes, even on randomly generated graphs. Thus it should work on contig variation graphs without added context. If this algorithm is applicable, then it points to cycle resolution being an MFAS problem. This would help formalize the problem of cycles in a contig variation graph.

3.4 Fine-tuning low-abundance node filtering

Low-abundance node filtering removes noise by discarding nodes whose abundance falls below a predefined minimum threshold. It is defined as a percentage of the largest abundance a node has in the graph. In previous work [11], this has been set to 0.5%. As of writing I could not find an explanation for why this percentage was chosen. Thus I assume this has not been researched yet.

4 Experimental Setup and Results

I will now evaluate the three strategies by their effect on performance. KLAE is used for every test needing to compare runtime and accuracy of assembling haplotypes. k is estimated using MPC with subpath constraints. All KLAE tests are run on the DHPC [4] compute node using 8 threads with 3968MB RAM each.

To assess KLAE's accuracy in assembling errors, genome fraction and assembly errors are used as metrics. Genome fraction is the percentage of reference genome base pairs that are covered by the solution haplotypes. It is defined as (aligned base pairs in the reference / reference genome length) $\times$ 100. Assembly errors are the percentage of base pairs that are either mismatches, indel errors or N's per aligned base. These values are generated using QUAST [8].

Every test uses datasets from the SAVAGE benchmark [1], namely 3-strain-ZIKV-20000x and 6-strain-poliovirus. The contig variation graph is built using an adjusted version of the pipeline, with contig filtering percentage set to 0.1%.

For cycle-resolution, four HIV graphs from [11] are used, as they contained cycles in the contig variation graph.

4.1 Abundance correction

I will test the effect of the implementation for abundance correction by first testing the effectiveness of a greedy heaviest path and a longest path as reference genome. This is done by comparing how close the coverage from the reference sequence is to the coverage of actual haplotypes in this dataset. Then the effectiveness of the alignment is evaluated. Finally, KLAE is performed on the graphs with and without abundance correction. The results are compared based on the runtime and accuracy of these results to see how weight correction has impacted the solution.

Greedy heaviest path and longest path work well for referencing a genome

To asses effectiveness of these paths, the coverage of the reference path is compared to the sum of the coverage of each ground-truth strain in the dataset. Dynamic Time Warping [13] (DTW) is used to compute a distance metric between the coverage from the reference strategy and the actual coverage of the solution strains.

Table 1 shows the DTW results. Both greedy heaviest path and longest path are a significantly better representation of coverage than the Uniform reference: the distances are on average 30 times smaller for ZIKV and 100 times smaller for poliovirus. Greedy heaviest path performs better for Poliovirus, but longest path performs better for ZIKV.

Strategy	ZIKV	Poliovirus
Uniform reference	390029	411182
Greedy heaviest path	14579	3270
Longest path	10870	4995

Table 1: DTW similarity score for different referencing strategies compared to the coverage of the solution haplotypes, a score closer to 0 means the reference better represents the actual coverage. Uniform reference has the mean actual coverage for each base pair. It serves as a baseline of similarity.

In the following experiments, greedy heaviest path will be used exclusively. This is arbitrarily chosen as there is no clear best strategy.

Mapping using BWA-mem2 alignment is not perfect

The mapping strategy will be evaluated by comparing it to the solution coverage. The effectiveness is characterized by how much the coverage of the solution strains matches with mapped abundance. Figure 3 shows that there is potential for this method, with regions where coverage and abundance are identical. However certain mappings generated are faulty, which can be seen by the spike in the mapped abundance graph around base-pair index 1800.

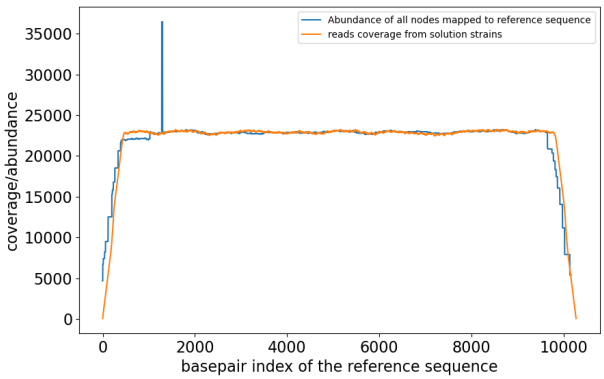

Figure 3: Node mapping compared to solution coverage for the 3-strain ZIKV dataset. The node mapping plot is calculated by adding its abundance to every index it is aligned to. Less difference in both plots indicates better mapping.

Scaling is sensitive to mapping errors

Using the created mapping, nodes are scaled to their associated coverage. From the ZIKV example in Figure 4, it can be seen that when coverage and abundance align, the rescaled abundance creates a flat line. The best improvement can be seen at the end. However when they are not aligned, the rescaled abundance does not improve. This can be seen at the very beginning where the abundance is relatively still low. It can even make the imbalance worse, when for example the abundance is high when the coverage is low.

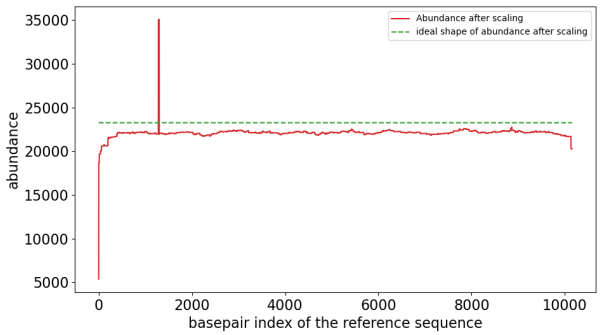

Figure 4: Result of rescaling abundance for the 3-strain ZIKV dataset. The rescaled node plot is calculated by adding its abundance to every index it is aligned to. A more straight line on the graph representing the abundance over the reference sequence is better.

K least absolute errors results

For both datasets the KLAE performance with and without abundance correction is shown in Table 2. The figure shows the average performance over 10 runs. The runtime measurements exclude the abundance correction step. This step was omitted to isolate the effect of abundance inaccuracies on solution runtime and accuracy.

It can be seen that abundance correction reduces runtime when the solution gets more complex. The Polio virus dataset is 15.8% faster using abundance correction. The ZIKV dataset did not have a performance impact and is even marginally slower. A reason for increased runtime could be errors in the mapping and rescaling, as shown in Figure 4.

At the same time the genome fraction and errors are not impacted, which means the solution still is a valid genome reassembly.

4.2 Performance impact low abundance node filtering

The next pre-processing strategy evaluates different configurations for the low abundance node filter. It is important to note that assessing performance of low-abundance node filtering on simulated datasets leads to incorrect conclusions. As simulated data may not represent real life scenarios. Thus tests are run with real sample data, obtained from the SAVAGE benchmarks dataset. The following cases are chosen [1]: 3-strain ZIKV and 6-strain poliovirus. I generated datasets with 30 different filter threshold percentages: between 0% to 3% with steps of 0.1%. KLAE is performed on each configuration comparing runtime and accuracy.

	Abundance correction	
	Without	With
3-strain ZIKV		
Assembly error (%)	0.0033	0.0033
Genome fraction (%)	99.029	99.029
Runtime (s)	0.87	0.94
6-strain Poliovirus		
Assembly error (%)	0.0987	0.0987
Genome fraction (%)	99.687	99.687
Runtime (s)	2102	1768

Table 2: Performance of running KLAE on both the 3-strain ZIKV and 6-strain Poliovirus dataset. Runtime and accuracy are displayed and compared to results without the abundance correction step. the statistics are the mean taken from 10 runs.

Adjusting the filter affects MPC results

During testing I found that the MPC results change when adjusting the low-abundance node filter. Figure 5 plots the MPC results for the different configuration. After 0.3% the k stabilizes, with k only being incorrect for poliovirus without subpath constraints.

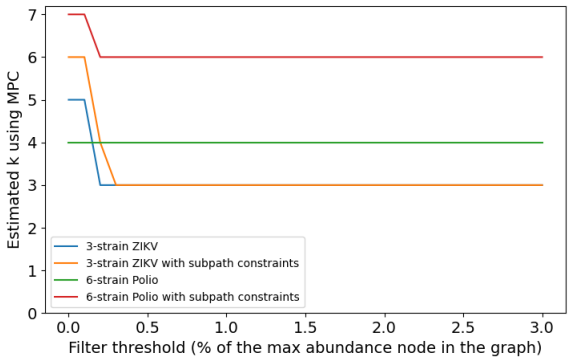

Figure 5: Result of running MPC on different abundance configurations. Both with and without subpath constraints are tested.

k least absolute error results

It is shown in Figure 6 that runtime decreases significantly for larger thresholds. The fastest configuration finishing 4 times faster than the original configuration of 0.5%. However, these fast configurations have a decrease in genome fraction of around 20%. Interestingly, assembly error decreases as the threshold is set higher, even when the genome fraction drops in quality. At the lower thresholds assembly error is around 0.15%, while the highest thresholds have an assembly error of 0.09%.

A middle ground between runtime and accuracy is found in the configurations ranging from 1.1% and 1.8%. Runtime is reduced by 25%, while genome fraction stays above 99%.

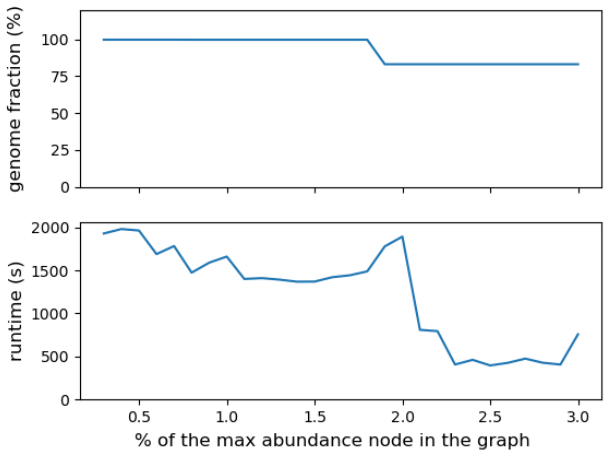

Figure 6: KLAE results on the poliovirus dataset, here 2 statistics are described: genome fraction (higher is better) and runtime (lower is better). The mean over 3 runs is used as values. Only shows results from configurations higher than 0.2%. Configurations below this percentage did not finish in time.

4.3 Cycle resolution

There are only datasets containing simple small cycles. Thus for this part of the research I created a method to introduce cycles into the graph artificially.

The first implemented approach uses a DFS to iterate through the graph, when visiting a node, there is a chance to create an edge to a random visited node. This could create non-cycles, as a node further down the graph may have already been visited. To verify if a cycle has been created, it is checked if a path exists from the already visited node to the currently selected node. The edge is removed again if no cycle was created. The generator has been configured to have a 0.5% chance to generate an edge at every unvisited node for the created datasets. As these cycle edges do not generate subpath information, it means that evaluation using KLAE cannot be done.

Results

Both methods of cycle resolution are compared on the different datasets. Figure 7 shows the results of both strategies.

CycleDFS nearly makes all correct cuts. However, the amount of incorrect cuts is in the worst case ten times more than the correct amount of cuts.

The H-Voting strategy fails to produce correct edge cuts. While the overall number of cuts remains low, none correspond to the true cut locations. This indicates that H-voting optimizes for a different problem and breaking cycles in a contig variation graph cannot be modeled as a MFAS problem.

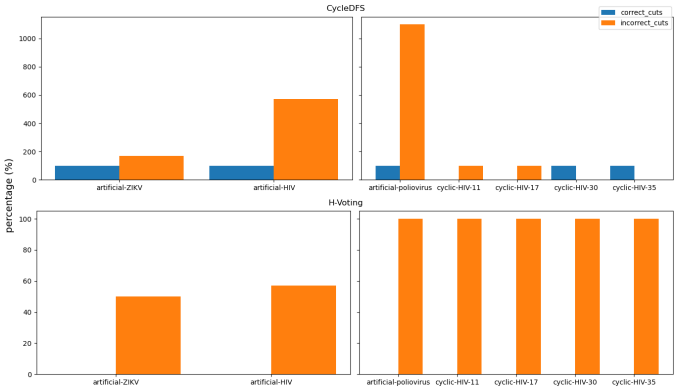

Figure 7: Result of CycleDFS cycle-resolution and H-voting. First column subplot shows for datasets with more than one cycle. Second column shows datasets with only 1 cycle. datasets with artificial in the name are created using cycle generation. Cyclic already included cycles.

5 Responsible Research

5.1 Use of LLMs

ChatGPT was used as a feedback generator to improve writing and sentence structure. Prompts such as ”what is this abstract missing?” or ”can you point to all spelling and grammar mistakes?” are used for sections, receiving suggestions instead of only complete rewrites. For single sentences, prompts like ”This sentence feels awkward, how do I improve this?” are used.

ChatGPT was also used as a LaTeX formatter. It enerated pseudo code using source code as input. This created verbose pseudo code too large for a single column. Thus the output was manually adjusted and abbreviated for clarity and to ensure it represents the source code correctly. Additionally, table formatting was done with the request: ”give a LaTeX format table with columns: strategy, DTW cost and rows: Greedy heaviest path, longest path”. Values were manually inserted into these tables, ensuring validity of the data displayed.

5.2 Reproducibility

To ensure reproducibility of this research comments and README files are included to guide the user. Scripts are grouped by the tests they are associated with.

The source code used in this research is available on the EWI GitLab. Furthermore, code used for the H-voting algorithm is found in the repository associated with the paper for the algorithm. Links to these sources are included in Appendix A.

6 Conclusions and Discussion

This research evaluated three strategies with the goal of improving the contig variation graph for haplotype assembly. The strategies include correcting abundance based on coverage bias errors, finding an optimal threshold for low-abundance node filtering and resolving cycles automatically in the graph.

6.1 Abundance correction

Applying abundance correction based on coverage is a promising strategy. For the poliovirus dataset, runtime is reduced by 14.8% using abundance correction, without compromising assembly quality. However, for the ZIKV runtime on average was longer using abundance correction. This is likely caused by mapping errors from faulty alignments made by BWA-mem2, or off-by-one errors when extracting the node mapping from the alignment file.

This strategy is split into 3 parts: finding a reference path in the contig variation graph, mapping the coverage of this path back to the rest of the graph and finally scaling each node abundance. Different strategies for each of these parts could lead to improved results in the correction of abundance. An example of another walk strategy could be using a known DNA genome of the virus. Of course, this takes away the *de novo* character of the MFD approach. Furthermore, other scaling methods might be less prone to errors between abundance and coverage.

Since only two datasets are evaluated, future work should analyze this method on a broad range of datasets to proof the general applicability and limitations of this strategy.

6.2 Low-abundance node filtering

Low-abundance node filtering is crucial for correct and efficient MFD solutions, as minimal node filtering leads to a higher MPC estimation. Results show that the node filter should be set to at least 0.3%. Furthermore, performance results on poliovirus datasets show that setting the filter up to 1.8% reduces runtime without compromising genome fraction. Another observation is that an increased filter threshold decreases the error rate of the assembly, indicating that indeed more erroneous nodes are removed.

As only one dataset has been analyzed, future work should test the recommended thresholds on many different datasets to verify its effectiveness. Future work should also explore other ways of defining the threshold, for example as a percentage of coverage depth.

6.3 Cycle resolution

Both proposed cycle resolution strategies make excessive incorrect edge cuts or insufficient correct edge cuts, which would severely impact MFD performance. H-voting results suggest that framing cycle resolution as an MFAS problem is inappropriate for contig variation graphs. This is inferred from the fact that all the edge cuts H-voting are incorrect and makes less cuts than the amount of edges that were erroneous.

Other methods and adjustments for cycle resolution

Other edge cutting heuristics could improve both of the proposed strategies. Properties of the contig variation graph could be used in the future, like difference in node abundance.

The VG tool-set [5] has a command to remove cycles from a graph using a topological sort approximation. This method could already be capable of breaking up cycles for contig variation graphs.

Cycle resolution strategies were evaluated using datasets with cycles artificially introduced. This may have led to unrealistic graphs and the results should be interpreted with this in mind.

Cycles occurring naturally in data are assumed from [11], though the savage benchmark datasets did not include cycles. Finding real example data contain cycles in the contig variation graph is essential for further research into this topic.

6.4 Contig filtering is important

In this research, the contig filter works based on a threshold defined as a percentage of the maximum abundance of contigs. When using the SAVAGE dataset, filtering is a crucial step to get a feasible and quality assembly.

Future work should look into assessing options for filtering noisy contigs and finding an optimal configuration which improves performance for a wide range of datasets.

6.5 Performance comparison with VG-flow

VG-flow [2] is another tool used for haplotype assembly. The paper contains assembly benchmarks for the 6-strain poliovirus dataset, also used in this research.

When compared, KLAE performs better in terms of genome fraction achieving 99.7%, while VG-flow results get 90.2%. Furthermore VG-flow overestimates the amount of haplotypes, finding 12. However it has a lower assembly error of 0.036%, while KLAE has 0.099%.

The observed improvements could be caused by MFD having an advantage in haplotype assembly or by differences in the generation of contig variation graphs.

6.6 Final conclusions

This research shows the potential of abundance correction, at the same time displaying its current instability. It also reveals the importance of node filtering and an optimal configuration. This configuration should be further tested on a broad range of datasets. Finally, the proposed cycle resolution algorithms turn out to not be applicable to the problem of cycles in a contig variation graph.

A Links to repositories

Source code used in the project, under branch Senne-Drent: https://gitlab.ewi.tudelft.nl/jbaaijens/cse3000-viralmfd

Source code for the H-voting algorithm can be found at: https://github.com/zhenv5/breaking_cycles_in_noisy_hierarchies

References

[1] Jasmijn A. Baaijens, Amal Zine El Aabidine, Eric Rivals, and Alexander Schönhuth. De novo assembly of viral quasispecies using overlap graphs. *bioRxiv*, 2016.

[2] Jasmijn A. Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. *bioRxiv*, 2020.

[3] Nicolas L. Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature Biotechnology*, 34(5):525–527, May 2016.

[4] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

[5] Erik Garrison, Jouni Sirén, Adam Novak, Glenn Hickey, Jordan Eizenga, Eric Dawson, William Jones, Shilpa Garg, Charles Markello, Michael Lin, and Benedict Paten. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36:875–879, 08 2018.

[6] Andreas Grigorjew, Fernando H.C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu. Accelerating ilp solvers for minimum flow decompositions through search space and dimensionality reductions. volume 301, 2024. Cited by: 1.

[7] Mangesh Gupte, Pravin Shankar, Jing Li, S. Muthukrishnan, and Liviu Iftode. Finding hierarchy in directed online social networks. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 557–566, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 02 2013.

[9] Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *2012 Proceedings IEEE INFOCOM*, pages 828–836, 2012.

[10] Ralf Herbrich and Thore Graepel. Trueskill(tm): A bayesian skill rating system. Technical Report MSR-TR-2006-80, January 2006.

[11] R.R. Huizenga. Minimum flow decomposition for haplotype-aware genome assembly. Master's thesis, TU Delft, 2025.

[12] Richard Karp. Reducibility among combinatorial problems. volume 40, pages 85–103, 01 1972.

[13] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[14] Jiankai Sun, Deepak Ajwani, Patrick K. Nicholson, Alessandra Sala, and Srinivasan Parthasarathy. Breaking cycles in noisy hierarchies. In *Proceedings of the 2017 ACM on Web Science Conference*, WebSci '17, page 151–160, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Md. Vasimuddin, Sanchit Misra, Heng Li, and Srinivas Aluru. Efficient architecture-aware acceleration of bwa-mem for multicore systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 314–324, 2019.