

Weight sets for paths in minimum flow decomposition

Finding good weight sets for paths in minimum flow decomposition for viral genome analysis

Agnese Elerte

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: Agnese Elerte
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

Finding new virus mutations is a valuable task, allowing us to more accurately predict further mutations and treat existing ones. In this paper, we explore the use of Minimum Flow Decomposition using Integer Linear Programming to reconstruct viral haplotypes. The runtime bottlenecks of this method are significant. Providing a set of weights for the ILP solver is a method to reduce the runtime. We evaluate the runtime and accuracy of the solver using 3 different weight generation methods. The methods are as follows: (1) sums of two powers of 2, (2) clustered weight values and their absolute differences, (3) clustered weight values and their minimum generating set. We evaluate the performance of each method with perfect haplotype graphs, HIV graphs, and HCV graphs. The results show that the cluster-based methods offer a significant reduction in the runtime, while sums of powers of 2 do not offer a significant runtime improvement. It is found that using a weight superset often leads to larger inaccuracies in the paths and their assigned weights. No significant correlation was found between the accuracy or size of the weight set and the runtime of the solver.

1 Introduction

Minimum Flow Decomposition is the problem of finding a set of weighted paths for a directed acyclic graph, so that for every node, the sum of weights of the paths going through it is equal to the weight of the node [5]. Flow decomposition is used in various network routing problems [12],[11], but in this paper, the focus is put on using MFD in viral genome analysis, more specifically, the assembling of viral genetic sequences.

The problem of assembling genetic sequences can be modeled as an MFD problem. The nodes in the graph represent short genetic sequences. An edge connects two nodes if their respective genetic sequences have been found next to each other in the reference data. The weight of a node indicates how many times the sequence appears in the data.

By solving this model, the final paths are longer genetic sequences (viral haplotypes), and their weights are the abundances of the haplotypes in the sample. Viral haplotypes are the mutated versions of a virus, where the differences between the original and the mutation are minor changes in the genetic sequence. By finding these new haplotypes, it is possible to track the evolution of a virus and develop treatment more effectively [8]. An example of a minumum flow decomposition is shown in Figure 1.

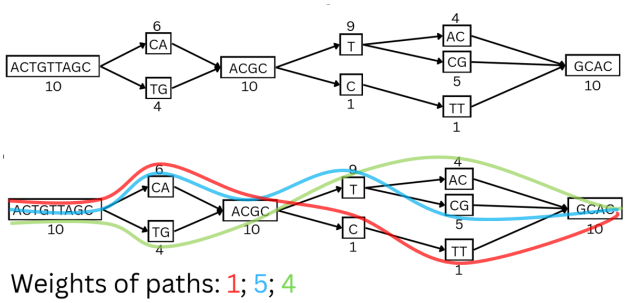


Figure 1: An example of a minimum flow decomposition of a graph. The nodes contain genetic sequence fragments. The MFD of this graph results in 3 paths that each reconstruct a longer genetic sequence.

The MFD problem is NP-hard [7], [6]. Recent research has shown that Integer Linear Programming (ILP) based solvers are a promising approach to solving MFD problems [8], but the solver's runtime is too high for complex graphs [3]. A suggested approach to reducing the runtime is to give the ILP solver a set of weights to choose from when assigning weights to the paths. By default, the solver is allowed to select any positive integer weight, which means that it has to consider a large set of different possible weight assignments. By giving a smaller set of possible weights, the amount of combinations of weights and paths is reduced, and the runtime is lowered.

Past studies have researched the effect that sets of weights have on the performance of the MFD ILP solution. In [8], using a set of weights, generated using k-means clustering, resulted in lower runtime for some MFD solutions. The same conclusion was reached in [4] and [9], using powers of 2 and node weights.

A comparison between the different weight methods has not been done yet, and there are possible weight selection methods that have not been explored. The goal of this research paper is to present multiple new or improved existing weight selection methods and compare their performance. In addition to this, it will be examined how different noise levels in node weights affect the performance of the solver.

1.1 Research Question

The research question that this paper will try to answer is *"How can we find a set of candidate weights that reduce the runtime bottlenecks of flow decomposition using integer linear programming for viral genome analysis?"*.

This research question is divided into the following subquestions:

1. What methods for selecting weight sets for MFD ILP have already been used or tested? What methods are suggested by previous research?
2. How does the selection of weight sets for MFD ILP affect the runtime and accuracy of the ILP solver?
3. How does the level of noise in node weights affect the runtime and accuracy of the ILP solver using different weight selection methods?

2 Background

2.1 Exact MFD and relaxing constraints

Minimum Flow Decomposition returns the minimal number of paths such that for every node, the sum of the weights of the paths going through that node is equal to the weight of the node. This requires the graph to be perfect, and noise in the node weights may cause the decomposition to be impossible. Since the graphs used for reconstructing haplotypes can contain inaccuracies and noise [8], this is a relevant problem.

This issue can be handled by relaxing the constraint of the flow equaling the weight of the node it is going through. Instead, a different objective function is used with the goal of minimizing the sum of weight differences for each node and the flow going through it. This approach is called k-least absolute errors and was used in [8].

K-least absolute errors requires the number of paths k to be defined. MPC (Minimum Path Cover) is a viable solution since its length is the lower bound estimate for the number of paths in an MFD solution [2]. The Minimum Path Cover is the set of paths that covers each node of the graph at least one time. As stated in [14], lower bounds for the number of paths that do not take into account the weights of the nodes may heavily underestimate the number of paths. The MPC approach of estimating the number of paths does not take weights into account, but by using subpath constraints (all subpaths also have to be covered by at least one path), the estimate is improved. This is based on [8], where it is stated that MPC with subpath constraints is a reliable estimate for the amount of paths in the graph, further supported by [15], where it was found that the decomposition accuracy increases when subpath constraints are used.

2.2 Weight sets

For MFD problems, multiple weight set generation methods have already been examined. One of the first ideas for weight sets to use in MFD problems was using the graph flow values [9]. This approach resulted in a better performance, with the unoptimized version of the algorithm on average taking 15% longer to finish.

In [8], a k-means clustering approach is used, using an elbow approach to estimate k. It was found that the weight distribution when using the cluster centers was not accurate, containing too many high weights. Due to this, the solver had to discard some of the paths, since no more weight could be assigned. The importance of the accuracy of the estimated weights was also reflected in the results - the performance of the algorithm was better if the estimated weights were close to the true weights. It was found that if the solver was provided with a perfect set of weights, it would have a significant improvement in the number of samples completed.

The approach of using powers of 2 together with node weights was examined in [4]. The results indicated an improvement in performance when using a weight superset compared to not using one. The accuracy of the solutions depended on the number of paths - samples with more paths had more accurate solutions. It is hypothesised that if the node weights added to the weight set are the same as for the true paths, the accuracy improves. It is noted that this approach is not optimal for all graphs and suggests using other sets of 2^i for any $i \in Z^+$.

3 Methodology

To answer the research question, multiple weight selection methods were tested: (1) powers of 2; (2) cluster with subtracting; (3) cluster with MinGenSet. The weight selection methods were used to generate the set of weights that were then input into the ILP solver.

We chose to use the k-least absolute errors approach in combination with minimum path cover to estimate the amount of paths. The solver used is from the FlowPaths[1] package since it includes implementations for k-least absolute errors and minimum path cover.

For clustering, KMeans [2] and Agglomerative Clustering [3] are used. For KMeans, an elbow approach is used to determine the number of clusters, based on the approach used in [8]. For Agglomerative Clustering, the distance threshold is used to determine the number of clusters. The threshold is set at 1% of the largest node weight. In the data graphs used in our experiment, any node that has a weight below 0.5% of the largest node weight of the graph is filtered out. Thus, we chose a similar approach to handle noise in the clustering process.

3.1 Powers of 2

In [4], powers of 2 with node weights are used. We aim to further test this approach by using the following equation for generating weights:

$$2^i + 2^j, \text{where } 2^i + 2^j < w_{max} \text{ and } i < j \qquad (1)$$

where w_{max} is the largest weight of any node in the graph.

The reason for including more weights is due to the distribution of larger weights. By using only powers of 2, the gaps between the larger weights would be too large to provide a diverse set of weights.

We do not include the node weights in this set, since this set includes more diverse weights for the graph than just using powers of 2.

3.2 Cluster with subtracting

In [8], the idea of clustering using K-means is introduced. While this approach performs reasonably well with simple graphs, more complex graphs introduce problems. If a graph has a path that has no node that is unique to this path, the simple clustering approach would not be able to extract the weight of the path.

Thus, we propose a new approach that tries to remove some unnecessary weights and add weights that could possibly be a path weight. The steps are as follows:

1. Cluster all node weights and extract the centers of these clusters as the base set;

[1] https://algbio.github.io/flowpaths/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

[3] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

2. Remove all weights that can be expressed as the sum of 2 other weights, since these weights are very likely to be for nodes that are included in multiple paths;
3. For all pairs of weights, calculate the absolute difference between them;
4. Cluster all of the absolute differences, extract the centers, and add the weights to the base set;

There may be graphs where multiple paths have the same weight. In this scenario, this weight generation method would create one cluster for both paths and thus the final set would include only one weight for both paths. To tackle this, the final set actually includes a duplicate of each weight generated in step 2.

3.3 Cluster with MinGenSet

The minimum generating set (MinGenSet) problem is an NP-hard problem [10] - given a set of integers s and an integer k, find the smallest set t such that all elements in s can be expressed as a sum of some elements in t and the sum of all elements in t is equal to k. This can be used to generate a set of weights if s is the set of cluster centers after clustering all node weights and k is the largest of the cluster centers.

Due to runtime constraints, this method will use a rounding approach where all values in s will be rounded using 1% of the mean weight value as the smallest unit. This way, the NP-hard problem can be solved faster while also retaining the original values of s. This method was found through testing.

Similar to clusters with subtracting, this method uses clustering, so we duplicate all weights generated for the final set of weights.

The length of the MinGenSet is also a lower bound for the amount of paths in the MFD. If the length of the solution set of MinGenSet is k, we must be able to find a set of $k + 1$ paths for the MFD problem [6]. We did not use this as a lower limit due to its sensitivity to noise - it would over-estimate the number of paths for many graphs.

3.4 Noise generation

The graphs for viral sequence assembly are expected to have some noise due to errors in the construction of the graph. For a weight generation method, it is important to take noise handling into account.

To examine the handling of noise with the different weight generation methods, the graphs will be introduced with noise. The noise will be generated for the perfect graphs, which originally do not have any noise in them.

The noise will be generated in the following way:

1. Generate an array of half the length of the entire node set that is filled with samples from a Poisson distribution with $\lambda = 1$.
2. Generate an array of half the length of the entire node set that is filled with samples from a Poisson distribution with $\lambda = 1$ and make all samples negative.
3. Append the two arrays and shuffle them.
4. For each of the node, adjust its weight by $0.01 * k * w$, where k is the Poisson distribution sample and w is the weight of the node.

To expand on noise further, another set of noisy graphs was generated with 0.03 in the place of 0.01 for step 4. In further analysis, this value is referred to as base noise.

4 Experimental Setup and Results

To answer the research questions, we chose to conduct experiments. The independent variable in these experiments is the weight generation algorithm. The dependent variables are the runtime, paths, and their assigned weights.

The following metrics are used for evaluating the accuracy of the haplotype assembly:

1. Number of estimated paths (k) - the total number of haplotypes calculated by the minimum path cover.
2. Number of paths that the estimated paths match to (k_{fin}) - the number of true haplotypes that the paths match to (sometimes smaller than k due to multiple paths matching to one haplotype).
3. Number of switch errors per haplotype (e_{sw}) - number of times the estimated paths switch from one reference haplotype to another.
4. Number of mismatches per 100 kbp (e_n) - number of mismatched nucleotides per 100,000 aligned nucleotides.
5. Weight error (e_w) - average error of the weight assigned to the haplotype when compared to the matched true path. Calculated by dividing the absolute difference between the generated weight and true weight with the true weight.

4.1 Experimental Setup

All experiments were ran using DelftBlue [1], using 16 CPUs with 3968MB of memory per CPU. All experiments are run for 24 hours.

The datasets used for tests are sourced from [8] and are available online[4]. All datasets are synthetic. The perfect graphs are graphs that have a perfect flow decomposition possible and have no noise. The HCV and HIV are simulated to reflect a more realistic scenario with noise. Further description of the datasets is available in [8].

Each experiment is set up to run the datasets in increasing difficulty (number of haplotypes) for each type (perfect graphs, HIV, HCV) - starting with the graphs containing lower amounts of haplotypes.

The Flowpaths package allows to specify the ILP solver, we chose to use Gurobi. For data analysis, Quast[5] was used.

4.2 Clustering methods

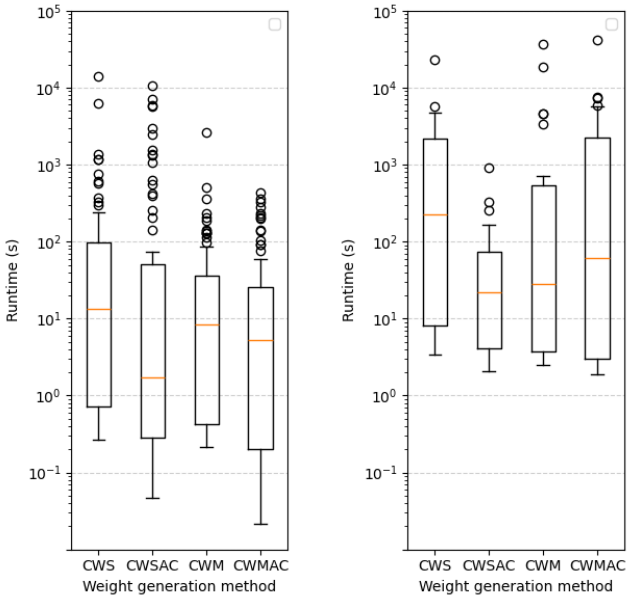
The cluster with subtracting and cluster with MinGenSet weight generation methods rely on a clustering method. The two methods tested were KMeans and Agglomerative clustering. We will evaluate these methods and choose one of the clustering methods for each of the weight generation methods.

[4] https://doi.org/10.5281/zenodo.16910556

[5] https://github.com/ablab/quast

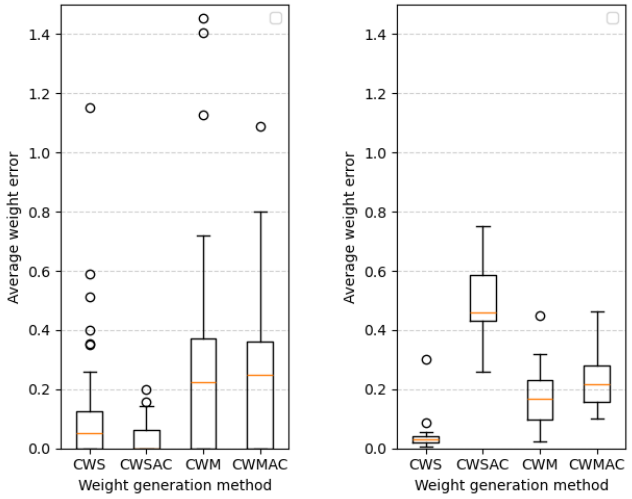
For this section, we refer to the following weight selection methods: CWS - Cluster with subtracting using KMeans clustering; CWSAC - Cluster with subtracting using Agglomerative clustering; CWM - Cluster with MinGenSet using KMeans clustering; CWMAC - Cluster with MinGenSet using Agglomerative clustering.

To compare the two methods, we will compare the weight errors of the generated weight sets and the runtime of completed graphs. In this case, weight errors are calculated by alligning most closely matching pairs from all of the weights generated and the true weights. The results of this comparison can be seen in Figures 2a, 2b, 3a, 3b. Results for HCV graphs are included in Appendix A, as they exhibit a very similar behavior to HIV graphs. For the average weight error graphs, there are a few values that are above the threshold of 1.5. We chose to omit these from the graph to visualize the general pattern better.



(a) Runtimes for perfect graphs (b) Runtimes for HIV graphs

Figure 2: Runtimes for perfect graphs and HIV graph, using clustering based methods. The following labels are used: CWS (Cluster with subtracting using KMeans), CWSAC (Cluster with subtracting using Agglomerative Clustering), CWM (Cluster with MinGenSet using KMeans), CWMAC (Cluster with MinGenSet using Agglomerative Clustering)



(a) Average weight errors for perfect graphs. (b) Average weight errors for HIV graphs

Figure 3: Average weight errors for perfect graphs and HIV graph, using clustering-based methods. The following labels are used: CWS (Cluster with subtracting using KMeans), CWSAC (Cluster with subtracting using Agglomerative Clustering), CWM (Cluster with MinGenSet using KMeans), CWMAC (Cluster with MinGenSet using Agglomerative Clustering)

For runtimes, we can see that the CWSAC is significantly faster on the HIV graphs and somewhat faster on the perfect graphs, compared to CWS. Between the CWM and CWMAC methods, CWM is slightly slower on perfect graphs, but faster on HIV graphs.

The weight error rates also have contradictory results, but the error for CWSAC is significantly worse than CWS for HIV graphs. Due to this we decided to go with CWS instead of CWSAC. Between, CWM and CWMAC, the CWM has lower weight error and a similar runtime, thus we chose to go with CWM. For the rest of this paper, we will use cluster with subtracting using KMeans clustering and cluster with MinGenSet using KMeans clustering. They will be refered to without specifying the KMeans clustering method

4.3 Runtime results

The runtime of the solver in combination with the different weight generation algorithms can be seen in Figure 4. We can see that for the completed samples, the runtime using cluster-based methods is significantly better for graphs for which the number of expected haplotypes is 6 or above.

For the more complex HIV and HCV graphs, we compare samples completed instead of the runtime, visible in Figure 5. It is important to point out that if the solver is stuck on a specific sample, this would mean that the samples that are scheduled after this sample would not run. The ratio of finished samples is taking into account all samples, including ones that were never ran. We further discuss this in subsection 6.4. Nonetheless, we can see that the cluster-based methods do not get stuck on any graphs, unlike the powers of 2 approach and no weights.

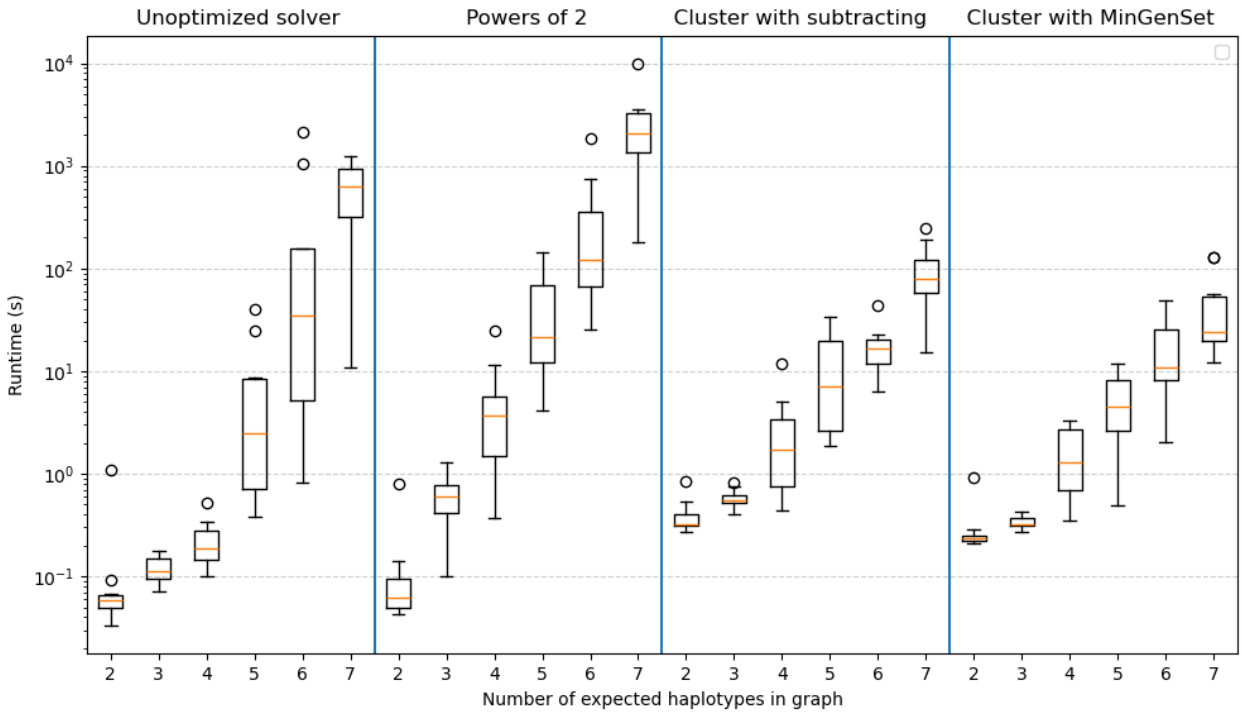


Figure 4: Runtime of the solver in combination with the different weight algorithms. The runtimes are grouped based on the number of expected haploypes (paths) in the graph. The data used is the perfect graphs.

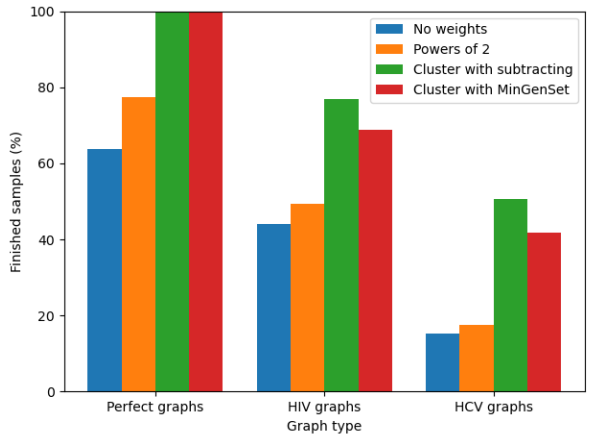


Figure 5: Percentage of completed samples for each graph type while using different weight generation methods. The data used are the perfect, HIV and HCV graphs.

4.4 Path accuracy results

The accuracy of estimated paths is done using the ground truth haplotypes used to generate the graphs. The resulting metrics for a set of perfect, HIV and HCV graphs are visible in Table 1.

	k	k_{fin}	e_{sw}	e_n	e_w
3-strain perfect graphs (10 samples)					
No weights	3.00	3.00	0.00	21.67	0.00
Powers of 2		3.00	0.00	25.00	0.25
CWS (KMeans)		3.00	0.00	40.00	0.01
CWM (KMeans)		3.00	0.00	20.00	0.00
6-strain perfect graphs (9 samples)					
No weights	5.44	5.22	0.00	100.76	0.45
Powers of 2		4.67	0.00	233.71	0.69
CWS (KMeans)		5.11	0.00	237.24	0.61
CWM (KMeans)		4.00	0.00	401.32	0.90
3-strain HIV graphs (6 samples)					
No weights	3.17	3.00	0.00	4.73	0.08
Powers of 2		3.00	0.00	4.73	0.13
CWS (KMeans)		3.00	0.00	3.68	0.13
CWM (KMeans)		3.00	0.00	3.68	0.35
3-strain perfect graphs (5 samples)					
No weights	3.00	3.00	0.00	5.03	0.02
Powers of 2		3.00	0.00	5.03	0.10
CWS (KMeans)		3.00	0.00	5.03	0.03
CWM (KMeans)		3.00	0.00	5.03	0.23

Table 1: Assembly accuracy for a set of graphs. Each section contains the average values for all graphs in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

We can see that for the larger haplotype number perfect graphs, the estimated number of paths k is lower than the true number. Additionally, using a weight generation method decreases the value for k_{fin}, meaning that some paths are aligned to the same true path. For both the number of mismatches e_n and weight errors e_w, we can see that they both increase with the addition of a weight set.

4.5 Weight set accuracy and influence on runtime

We first examine the overall accuracies of the weight sets generated by calculating the average weight errors. The weight error for these results is calculated without taking the minimum flow decomposition into account. The full set of weights is matched to the true weights based on the best matching pairs. The results can be seen in Table 2. We can see that the weight error for both cluster-based methods increases with increasing number of true paths, while the opposite happens for powers of 2.

n	e_{wpow2}	e_{wcws}	e_{wcwm}
2	0.21	0.00	0.00
3	0.16	0.01	0.0
4	0.13	0.04	0.16
5	0.08	0.04	0.12
6	0.09	0.09	0.57
7	0.07	0.09	0.32
8	0.08	0.25	0.45
9	-	0.27	2.18

Table 2: The average weight error using the weight generation methods grouped by number of true paths in a graph. Data used is perfect graphs. n is the amount of true paths in the sample, e_{wpow2} is the weight error for powers of 2, e_{wcws} is the weight error for cluster with subtracting, e_{wcwm} is the weight error for cluster with MinGenSet.

To examine the influence of the weight set on the runtime of the solver, we use Pearson's coefficient. The coefficient is calculated between the runtime and different attributes of the weight set. The results for perfect and HIV graphs can be seen in Tables 3a and 3b, with HCV results in Appendix C as they closely resemble the results for HIV graphs.

Attribute	Pearson's coefficient
e_w	-0.027
$len(w)$	0.511
$len(w) - k$	0.472

(a) Perfect graphs

Attribute	Pearson's coefficient
e_w	-0.160
$len(w)$	-0.035
$len(w) - k$	-0.051

(b) HIV graphs

Table 3: Pearson's coefficients for weight set attributes and runtimes of the solver for perfect graphs and HIV graphs. e_w is the average weight error, $len(w)$ is the size of the weight set, k is the number of paths in the solution.

These results reveal that for perfect graphs, runtime has a correlation with the size of the weight set. For HIV graphs, the runtime shows no significant correlation with the weight error or size of the weight set.

4.6 Noise handling

The runtime of the solver with the weight methods can be seen in Table 4. It can be seen that with increasing noise, the runtime increases for all methods except for powers of 2.

BN	t_{nw}	t_{pow2}	t_{CWS}	t_{CWM}
0%	96.26	116.65	11.17	5.62
1%	155.86	83.49	10.81	9.52
3%	125.50	52.66	17.26	10.53

Table 4: Average runtimes for perfect graphs with varying noise levels. BN is the level of base noise. t_{nw} is the average runtime with no$_w eights$, t_{pow2} is the average runtime with powers of 2, t_{CWS} is the average runtime with cluster with subtracting, t_{CWM} is the average runtime with cluster with MinGenSet.

We also examine the accuracy of the assembled paths in Table 5. It can be seen that the number of mismatches e_n increases with the increasing noise for all weight generation methods. There is no significant increase in the weight error e_w. Some fluctuation can be seen in the k_{fin} value, but no consistent pattern can be seen.

	k	k_{fin}	e_{sw}	e_n	e_w
5-strain perfect graphs (10 samples)					
No weights	4.60	4.50	0.00	122.75	0.05
Powers of 2		4.40	0.00	261.52	0.29
CWS (KMeans)		4.50	0.00	188.25	0.16
CWM (KMeans)		4.00	0.00	293.07	0.78
5-strain noisy graphs (1%) (10 samples)					
No weights	4.60	4.40	0.00	128.25	0.05
Powers of 2		4.30	0.00	258.77	0.26
CWS (KMeans)		4.40	0.00	212.27	0.33
CWM (KMeans)		3.90	0.00	340.51	0.70
5-strain noisy graphs (3%) (10 samples)					
No weights	4.60	4.60	0.00	165.25	0.07
Powers of 2		4.40	0.00	303.77	0.27
CWS (KMeans)		4.30	0.00	258.79	0.19
CWM (KMeans)		3.90	0.00	382.29	0.49

Table 5: Assembly accuracy for noisy graphs. The graphs used are perfect graphs with no noise, 1% base noise and 3% base noise. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

5 Responsible Research

5.1 Usage in viral genome analysis

It must be acknowledged that by introducing a weight superset, the accuracy of the result may be affected. The accuracy of weights can affect the genomes reconstructed and may provide false or inaccurate haplotypes. If a weight generation method is used in a real-life scenario, this risk must be taken into account.

5.2 Reproducibility

All the results are reproducible by running the scripts in the codebase for this project[6]. The Delft Blue cluster should be used to ensure the same environment. All scripts used for analysing the results are also available in the codebase.

5.3 Usage of LLMs

LLMs were used in the setup of the DelftBlue experiment to help understand the issues with various packages and their installation, as well as for generating LaTeX templates. LLMs were not used to generate research ideas, text or code.

6 Discussion

In this section, we will explain the results and compare them to previous research.

6.1 Runtime and accuracy changes

Consistent runtime improvements can be seen for the two cluster-based methods, with less significant and sometimes no improvement for the powers of 2 weight generation method.

A very interesting observation is that using the powers of 2 weight generation method leads to larger runtime than using no weights for some samples. This is likely due to the powers of 2 weight set being significantly larger than the amount of paths. When compared to results in [4], we do not find the same runtime or accuracy improvements. This likely means that the weight set offers significantly better performance if the node weights are added to the weight set, agreeing with the hypothesis in [4].

The accuracy of the paths when compared to the pre-known paths shows that for the perfect graphs, the accuracy is almost always higher when the solver is ran with no weight set than with a weight set. This is expected, because as weights are added, the algorithm is more restricted, likely causing it to choose some less accurate paths. As expected, the accuracy decreases for all weight selection methods (and the unoptimized solver) as the number of true haplotypes increases, which matches other experiments [13].

We can observe an interesting behavior when looking at the k and k_{fin} values in Table 6. For the 6-strain perfect graphs, the k value is already below the expected. Nonetheless, the solver assigns multiple constructed haplotypes to one expected haplotype ($k > k_{fin}$). This is likely caused by an inaccurate weight estimation for a path. If an accurate enough weight is not present in the set, the solver may try to assign two paths to the same true path if their sum of weights is close to the weight of the path.

Finally, for all samples, the number of switch errors is 0. This is likely due to the Quast settings not being fit for the data we used. Using MetaQUAST and editing the acceptance criteria for a switch error might find switch errors in the samples. This was not done due to time constraints.

6.2 Weight set accuracy

In [8], it is stated that the better the estimation of the weights, the faster the runtime. The results in this paper do not show the same pattern, as the more accurate powers of 2 weights result in a larger runtime for perfect graphs. This could be explained by the size of the weight sets - the powers of 2 weight sets have more accurate weights estimated, but it often has a much larger set of weights, increasing the runtime.

This is also confirmed by the results using Pearson's coefficient, which showed a moderate relationship between the length of the weight set and the runtime, and a very weak relationship between the average weight error of the weight set and the runtime for perfect graphs. For HIV and HCV graphs, there is no significant correlation found.

6.3 Noisy graphs

The runtime for noisy graphs increases for the unoptimized solver as well as both of the cluster-based methods. An unexpected result is that using powers of 2 resulted in a lower runtime for noisy graphs than for graphs with no noise. Due to the powers of 2 set being independent of the graph, this decrease in runtime must be caused by a coincidentally better weight accuracy for the noisy graphs.

We can also see that the number of mismatches per 100 kbp e_n increases as the noise increases. This indicates that the noisiness of the graph does affect the accuracy of the paths in addition to their assigned weights.

6.4 Experimental setup

The setup of the experiment was done in a way that does not allow the solver to run later experiments if it gets stuck on an earlier graph. This is not optimal, since the solver may still be able to solve some of the later graphs.

The improvement we suggest is that for each haplotype count, the experiments should be allowed to run for 24 hours. This was not done due to time constraints.

7 Conclusions

Using weight sets for paths in Minimum Flow Decomposition is a promising approach to reducing the runtime of ILP solvers. We conclude that there can be significant runtime improvements, but reduced accuracy for paths and their assigned weights. Using a sum of two powers of 2 was found to be an inefficient method. The most promising methods explored are based on the node weights and clustering of them. It was also found that there was no consistent relationship between the runtime of the solver and the average error rate or size of the weight set provided to the solver.

[6] https://github.com/aelerte/weight_generation_for_MFD

7.1 Future recomendations

We recommend looking into further optimising the pre-existing weight generation methods and exploring new ones.

For cluster-based methods, we recommend looking further into different clustering methods in addition to refining the Agglomerative Clustering approach. It is important to explore how different clustering methods handle varying levels of noise. For Agglomerative Clustering, it would be useful to further optimize the distance threshold value to make the clustering more accurate. We also recommend further looking into how many duplicate weights should be added to the set or how to handle cases where multiple paths have the same weight.

Furthermore, we recommend looking into heuristics for the weight selection. For example, providing a set of weights to be assigned to some paths and allowing other paths to be assigned weights by the algorithm. Another possible method would be to iteratively remove the highest-weight path for some number of paths and use the solver to find the remaining paths.

References

[1] Delft High Performance Computing Centre (DHPC). DelftBlue Supercomputer (Phase 2), 2024.

[2] Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams. Width Helps and Hinders Splitting Flows. *ACM Trans. Algorithms*, 20(2):13:1–13:20, March 2024.

[3] Fernando H. C. Dias, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. Fast, Flexible, and Exact Minimum Flow Decompositions via ILP. volume 13278, pages 230–245. 2022. arXiv:2201.10923 [q-bio].

[4] 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. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms (SEA 2024)*, volume 301 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[5] Andreas Grigorjew, Wanchote Jiamjitrak, Brendan Mumey, and Alexandru I. Tomescu. Parameterised Approximation and Complexity of Minimum Flow Decompositions, October 2024. arXiv:2409.20278 [cs].

[6] Andreas Grigorjew, Wanchote Jiamjitrak, Brendan Mumey, and Alexandru I. Tomescu. Width Parameters for Minimum Flow Decomposition, November 2025. arXiv:2409.20278 [cs].

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

[8] Rael Rasa Huizenga. *Minimum Flow Decomposition for Haplotype-Aware Genome Assembly*. PhD thesis, 2025.

[9] Kyle Kloster, Philipp Kuinke, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D. Sullivan, and Andrew van der Poel. A practical fpt algorithm for Flow Decomposition and transcript assembly, August 2017. arXiv:1706.07851 [cs].

[10] Manuel Lozano, Manuel Laguna, Rafael Martí, Francisco J. Rodríguez, and Carlos García-Martínez. A genetic algorithm for the minimum generating set problem. *Applied Soft Computing*, 48:254–264, November 2016.

[11] Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity Balancing Path Flow Decomposition and Routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, December 2015.

[12] Nils Olsen, Natalia Kliewer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time–space network models. *Central European Journal of Operations Research*, 30(3):883–919, September 2022.

[13] Mingfu Shao and Carl Kingsford. Theory and A Heuristic for the Minimum Path Flow Decomposition Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, March 2019.

[14] B. Vatinlen, F. Chauvet, P. Chrétienne, and P. Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, March 2008.

[15] Lucia Williams, Alexandru I. Tomescu, and Brendan Mumey. Flow Decomposition With Subpath Constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, January 2023.

A Clustering methods

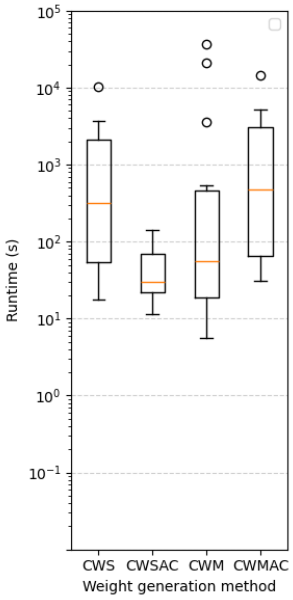


Figure 6: Runtime for HCV graphs.

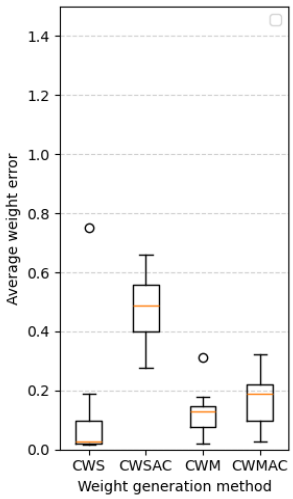


Figure 7: Weight error rates for HCV graphs

B Path and weight accuracy

B.1 Perfect graphs

	k	k_{fin}	e_{sw}	e_n	e_w
No weights	2.00	2.00	0.00	44.44	0.00
Powers of 2		2.00	0.00	50.00	0.25
CWS (KMeans)		2.00	0.00	38.89	0.00
CWS (AC)		2.00	0.00	55.56	0.00
CWMGS (KMeans)		2.00	0.00	16.67	0.00
CWMGS (AC)		2.00	0.00	66.67	0.11
3-strain perfect graphs (10 samples)					
No weights	3.00	3.00	0.00	21.67	0.00
Powers of 2		3.00	0.00	25.00	0.25
CWS (KMeans)		3.00	0.00	40.00	0.01
CWS (AC)		3.00	0.00	20.00	0.01
CWMGS (KMeans)		3.00	0.00	20.00	0.00
CWMGS (AC)		3.00	0.00	20.00	0.00
4-strain perfect graphs (10 samples)					
No weights	3.90	3.90	0.00	25.42	0.11
Powers of 2		3.40	0.00	145.83	0.56
CWS (KMeans)		3.80	0.00	97.92	0.18
CWS (AC)		3.90	0.00	43.75	0.09
CWMGS (KMeans)		3.60	0.00	180.83	0.54
CWMGS (AC)		3.70	0.00	161.25	0.53
5-strain perfect graphs (10 samples)					
No weights	4.60	4.50	0.00	122.75	0.05
Powers of 2		4.40	0.00	261.52	0.29
CWS (KMeans)		4.50	0.00	188.25	0.16
CWS (AC)		4.00	0.00	220.50	0.32
CWMGS (KMeans)		4.00	0.00	293.07	0.78
CWMGS (AC)		4.00	0.00	310.76	0.54
6-strain perfect graphs (9 samples)					
No weights	5.44	5.22	0.00	100.76	0.45
Powers of 2		4.67	0.00	233.71	0.69
CWS (KMeans)		5.11	0.00	237.24	0.61
CWS (AC)		5.00	0.00	212.04	0.59
CWMGS (KMeans)		4.00	0.00	401.32	0.90
CWMGS (AC)		4.33	0.00	408.74	1.10
7-strain perfect graphs (2 samples)					
No weights	6.00	6.00	0.00	157.15	0.10
Powers of 2		5.00	0.00	275.72	0.25
CWS (KMeans)		6.00	0.00	268.57	0.25
CWS (AC)		4.50	0.00	409.38	0.58
CWMGS (KMeans)		5.00	0.00	561.64	0.71
CWMGS (AC)		4.50	0.00	588.57	1.08

Table 6: Assembly accuracy for perfect graphs. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

	k	k_{fin}	e_{sw}	e_n	e_w
8-strain perfect graphs (10 samples)					
CWS (KMeans)	6.40	6.00	0.00	388.27	1.07
CWS (AC)		5.80	0.00	359.64	0.64
CWMGS (KMeans)		5.20	0.00	577.81	1.35
CWMGS (AC)		5.40	0.00	612.54	1.54
9-strain perfect graphs (10 samples)					
CWS (KMeans)	7.20	6.20	0.00	416.03	2.26
CWS (AC)		5.80	0.00	434.92	3.91
CWMGS (KMeans)		6.10	0.00	673.68	3.52
CWMGS (AC)		6.10	0.00	701.96	4.14

Table 7: Assembly accuracy for perfect graphs. Each section contains the average values for all samples in the set when compared to the true haplotypes. No weights and powers of 2 results are omitted due to them not having any results for these graphs. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

B.2 HIV graphs

	k	k_{fin}	e_{sw}	e_n	e_w
2-strain HIV graphs (9 samples)					
No weights	2.22	2.00	0.00	147.75	0.12
Powers of 2		2.00	0.00	66.50	0.23
CWS (KMeans)		2.00	0.00	88.28	0.36
CWS (AC)		2.00	0.00	96.47	0.56
CWMGS (KMeans)		2.00	0.00	15.30	0.96
CWMGS (AC)		2.00	0.00	119.71	0.81
3-strain HIV graphs (6 samples)					
No weights	3.17	3.00	0.00	4.73	0.08
Powers of 2		3.00	0.00	4.73	0.13
CWS (KMeans)		3.00	0.00	3.68	0.13
CWS (AC)		3.00	0.00	23.57	0.49
CWMGS (KMeans)		3.00	0.00	3.68	0.35
CWMGS (AC)		3.00	0.00	46.80	0.63
4-strain HIV graphs (7 samples)					
CWS (KMeans)	4.86	4.00	0.00	332.31	0.21
CWS (AC)		4.00	0.00	675.16	0.59
CWMGS (KMeans)		4.00	0.00	412.18	0.24
CWMGS (AC)		4.00	0.00	340.78	0.24

Table 8: Assembly accuracy for HIV graphs. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

B.3 HCV graphs

	k	k_{fin}	e_{sw}	e_n	e_w
3-strain perfect graphs (5 samples)					
No weights	3.00	3.00	0.00	5.03	0.02
Powers of 2		3.00	0.00	5.03	0.10
CWS (KMeans)		3.00	0.00	5.03	0.03
CWS (AC)		3.00	0.00	5.03	0.52
CWMGS (KMeans)		3.00	0.00	5.03	0.23
CWMGS (AC)		3.00	0.00	5.03	0.52
4-strain perfect graphs (4 samples)					
CWS (KMeans)	5.00	4.00	0.00	468.06	0.24
CWS (AC)		4.00	0.00	413.14	0.55
CWMGS (KMeans)		4.00	0.00	347.99	0.49
CWMGS (AC)		4.00	0.00	261.90	0.29

Table 9: Assembly accuracy for HCV graphs. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

C Pearson's coefficients for HCV

This section contains the Pearson's coefficients between runtime of a graph and various attributes of the weight set. Each type of graph had these coefficients calculated separately.

C.1 HCV graphs

Value to compare to	Pearson's coefficient
e_w	-0.226
$len(w)$	-0.089
$len(w) - k$	-0.099

Table 10: Pearson's coefficients for HCV graphs

D Path independent weight errors

This section contains the weight errors calculated independent of the path assignments.

n	e_{wpow2}	e_{wcws}	e_{wcwm}
2	0.05	0.06	0.25
3	0.05	0.03	0.10
4	-	0.04	0.13

Table 11: Average weight error for HIV graphs per haplotype number. n is the number of haploypes, e_{wpow2} is the average weight error for powers of 2, e_{wcws} is the average weight error for cluster with subtracting, e_{wcwm} is the average weight error for cluster with MinGenSet.

n	e_{wpow2}	e_{wcws}	e_{wcwm}
3	0.05	0.10	0.10
4	-	0.05	0.12
5	-	0.03	-

Table 12: Average weight error for HCV graphs per haplotype number. n is the number of haploypes, e_{wpow2} is the average weight error for powers of 2, e_{wcws} is the average weight error for cluster with subtracting, e_{wcwm} is the average weight error for cluster with MinGenSet.

E Noisy graph path and weight accuracies

	k	k_{fin}	e_{sw}	e_n	e_w
8-strain noisy graphs (1%) (10 samples)					
CWS (KMeans)	6.40	5.80	0.00	461.00	0.53
CWS (AC)		5.20	0.00	426.12	0.79
CWMGS (KMeans)		5.00	0.00	646.02	1.41
CWMGS (AC)		5.30	0.00	611.08	1.41
9-strain noisy graphs (1%) (8 samples)					
CWS (KMeans)	7.25	6.25	0.00	393.73	1.81
CWS (AC)		5.88	0.00	445.92	2.19
CWMGS (KMeans)		5.62	0.00	685.92	3.26
CWMGS (AC)		6.12	0.00	596.51	3.29

Table 14: Assembly accuracy for noisy graphs with 1% base noise. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

	k	k_{fin}	e_{sw}	e_n	e_w
2-strain noisy graphs (1%) (9 samples)					
No weights	2.00	2.00	0.00	44.44	0.00
Powers of 2		2.00	0.00	44.44	0.42
CWS (KMeans)		2.00	0.00	5.56	0.17
CWS (AC)		2.00	0.00	27.78	0.17
CWMGS (KMeans)		2.00	0.00	55.56	0.17
CWMGS (AC)		2.00	0.00	77.78	0.28
3-strain noisy graphs (1%) (10 samples)					
No weights	3.00	3.00	0.00	16.67	0.00
Powers of 2		3.00	0.00	40.00	0.34
CWS (KMeans)		3.00	0.00	18.33	0.14
CWS (AC)		3.00	0.00	1.67	0.14
CWMGS (KMeans)		3.00	0.00	35.00	0.24
CWMGS (AC)		3.00	0.00	35.00	0.25
4-strain noisy graphs (1%) (10 samples)					
No weights	3.90	3.90	0.00	27.50	0.11
Powers of 2		3.40	0.00	133.75	0.50
CWS (KMeans)		3.80	0.00	101.67	0.29
CWS (AC)		3.90	0.00	40.42	0.13
CWMGS (KMeans)		3.50	0.00	199.17	0.50
CWMGS (AC)		3.50	0.00	220.00	0.86
5-strain noisy graphs (1%) (10 samples)					
No weights	4.60	4.40	0.00	128.25	0.05
Powers of 2		4.30	0.00	258.77	0.26
CWS (KMeans)		4.40	0.00	212.27	0.33
CWS (AC)		4.10	0.00	269.06	0.31
CWMGS (KMeans)		3.90	0.00	340.51	0.70
CWMGS (AC)		4.10	0.00	375.29	0.84
6-strain noisy graphs (1%) (8 samples)					
No weights	5.38	5.25	0.00	176.48	0.47
Powers of 2		4.75	0.00	230.65	0.75
CWS (KMeans)		4.88	0.00	282.30	0.71
CWS (AC)		4.25	0.00	300.02	1.06
CWMGS (KMeans)		4.00	0.00	470.88	1.37
CWMGS (AC)		4.75	0.00	356.90	0.68
7-strain noisy graphs (1%) (1 samples)					
No weights	5.00	5.00	0.00	330.10	1.10
Powers of 2		4.00	0.00	360.11	0.60
CWS (KMeans)		5.00	0.00	330.10	1.15
CWS (AC)		5.00	0.00	230.14	0.59
CWMGS (KMeans)		4.00	0.00	460.28	1.48
CWMGS (AC)		4.00	0.00	420.25	0.73

Table 13: Assembly accuracy for noisy graphs with 1% base noise. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

	k	k_{fin}	e_{sw}	e_n	e_w
2-strain noisy graphs (3%) (9 samples)					
No weights	2.00	2.00	0.00	61.11	0.00
Powers of 2		2.00	0.00	72.22	0.41
CWS (KMeans)		2.00	0.00	22.22	0.17
CWS (AC)		2.00	0.00	66.67	0.17
CWMGS (KMeans)		2.00	0.00	33.33	0.20
CWMGS (AC)		2.00	0.00	72.22	0.20
3-strain noisy graphs (3%) (10 samples)					
No weights	3.00	3.00	0.00	30.00	0.00
Powers of 2		3.00	0.00	46.67	0.34
CWS (KMeans)		3.00	0.00	36.67	0.16
CWS (AC)		3.00	0.00	76.69	0.33
CWMGS (KMeans)		3.00	0.00	30.00	0.20
CWMGS (AC)		3.00	0.00	103.33	0.34
4-strain noisy graphs (3%) (10 samples)					
No weights	3.90	3.90	0.00	53.33	0.11
Powers of 2		3.60	0.00	165.42	0.40
CWS (KMeans)		3.80	0.00	137.08	0.24
CWS (AC)		3.40	0.00	150.42	0.57
CWMGS (KMeans)		3.50	0.00	189.59	0.49
CWMGS (AC)		3.40	0.00	214.17	0.71
5-strain noisy graphs (3%) (10 samples)					
No weights	4.60	4.60	0.00	165.25	0.07
Powers of 2		4.40	0.00	303.77	0.27
CWS (KMeans)		4.30	0.00	258.79	0.19
CWS (AC)		3.90	0.00	315.79	0.86
CWMGS (KMeans)		3.90	0.00	382.29	0.49
CWMGS (AC)		4.10	0.00	371.30	0.69
6-strain noisy graphs (3%) (7 samples)					
No weights	5.29	5.29	0.00	236.22	0.27
Powers of 2		4.86	0.00	265.26	0.62
CWS (KMeans)		5.00	0.00	327.40	0.66
CWS (AC)		4.43	0.00	324.54	0.78
CWMGS (KMeans)		5.00	0.00	352.88	1.01
CWMGS (AC)		4.29	0.00	315.75	1.18

Table 15: Assembly accuracy for noisy graphs with 3% base noise. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.

	k	k_{fin}	e_{sw}	e_n	e_w
7-strain noisy graphs (3%) (10 samples)					
Powers of 2	5.90	5.10	0.00	372.41	1.37
CWS (KMeans)		4.90	0.00	476.45	1.77
CWS (AC)		4.50	0.00	399.68	1.18
CWMGS (KMeans)		4.70	0.00	556.61	2.71
CWMGS (AC)		4.30	0.00	582.83	2.18
8-strain noisy graphs (3%) (10 samples)					
Powers of 2	6.40	5.70	0.00	457.63	0.61
CWS (KMeans)		5.80	0.00	460.73	0.48
CWS (AC)		5.10	0.00	518.09	2.02
CWMGS (KMeans)		5.50	0.00	597.28	1.53
CWMGS (AC)		5.00	0.00	625.24	1.36
9-strain noisy graphs (3%) (2 samples)					
Powers of 2	7.50	6.50	0.00	403.07	0.88
CWS (KMeans)		7.00	0.00	459.97	0.81
CWS (AC)		5.50	0.00	414.01	3.59
CWMGS (KMeans)		6.00	0.00	630.93	1.24
CWMGS (AC)		5.50	0.00	530.90	1.31

Table 16: Assembly accuracy for noisy graphs with base noise of 3%. Each section contains the average values for all samples in the set when compared to the true haplotypes. k stands for the number of estimated paths, k_{fin} is the number of unique true paths the paths align to, e_{sw} is the number of switch errors, e_n is the number of mismatches per 100 kbp, e_w is the average weight error.