

Subpath constraints in ILP-based Minimum Flow Decomposition to improve strain-aware viral genome assembly

Jona Bedaux[1]

Supervisor: Jasmijn Baaijens[1]

[1]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: Jona Bedaux
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) has the potential to improve strain-aware full genome assembly from contig variation graphs (CVGs) by replacing heuristics for path extraction [1]. Subpath constraints improve convergence time in the ILP formulation for MFD [2] and help strain-aware viral genome assemblers build a more accurate set of haplotype genomes from contig variation graphs [3]. In this paper, we mathematically formalize four alternative formulations for such subpath constraints and we implement these in a contribution to the open source flow decomposition toolkit *flowpaths* [4]. We evaluate the effects of these formulations on the runtime by benchmarking the decomposition process on a dataset of CVGs built from assembled HIV reads [1], and a dataset we generate ourselves, consisting of contig variation graphs with intentionally misassembled contigs to evaluate performance under noisy conditions.

We conclude that three of the four formulations tested affect the runtime negatively compared to the baseline. The last formulation is not usable in its current form due to issues with infeasibility, but does show potential. The CVGs that are feasible under this formulation converge quicker than under any other formulation. Therefore we recommend further research be done into improving feasibility for this subpath constraints formulation.

1 Introduction

RNA viruses mutate spontaneously at an extremely high rate within the human body [5]. Particularly, RNA viruses like HIV and HCV boast high mutation rates [6], [7] and cause chronic infection, which can lead to many genetically distinct viral strains coexisting in an infected patient [8]. This genetic diversity can contribute to chronic persistence of the infection [9], hampers the immune system's ability to fight back by it rapidly mutating its surface antigens [10], and is associated with a reduced efficacy for certain treatments [11].

Thus, a single genome is not enough to capture the virus in full. A viral population in an infection can be described as a quasispecies [12], consisting of a swarm of similar but not genetically identical viral particles. Each genetic variant within the quasispecies is called a haplotype. Tracking these haplotypes is essential to keep treatment targeted and effective.

It is generally very difficult to accurately assemble the genomes of separate haplotypes, as it is not possible to separate the haplotypes when sequencing. Sequencing technology is able to extract and read small portions of the target genome, referred to as *reads*. However, when sequencing a quasispecies, the sequencer cannot determine which haplotype a particular read originates from. We are thus left with a large set of pieces from all different haplotypes in the sample. The process of reconstructing multiple haplotypes from these mixed reads is called strain-aware full-genome assembly. As an additional complication, these reads contain errors – though their prevalence depends on the sequencing technology used. Third-generation sequencing produces longer reads but is more error-prone [13], while next-generation sequencing outputs shorter and more accurate reads [14]. This research assumes reads are generated by next-generation sequencing.

The first step of genome assembly is to combine overlapping similar reads into larger sequences called contigs. Contig assemblers are also tasked with comparing similar reads to ensure that the constructed contigs are error-free. Strain-aware contig assembly is similar, but special care must be taken to ensure contigs from different haplotypes are not erroneously joined together, requiring specialized strain-aware assemblers [15], [16], [17]. It can benefit assembly if a reference strain similar to the quasispecies is known, but we assume this is not the case and assembly has to be performed *de novo*.

One way to proceed is to construct a variation graph [18] from the contigs. This is a data structure that encodes the possible genomes that can be constructed by combining overlapping contigs. This *contig variation graph* (CVG) is then weighted according to the abundance of the subsequences in the reads. Finding a full set of weighted paths in the CVG that together cover the entire flow of the CVG constitutes finding a set of haplotypes that could generate the observed reads. To make sure the contigs are not broken up in the final genomes, the paths in this decomposition can be constrained to include the subpath laid out by every contig at least once. These subpath constraints tend to improve the assembly quality [3] and are the subject of our research.

Decomposing this flow in a minimal number of paths is provably difficult [19], and solving Minimum Flow Decomposition (MFD) on the CVG in full has been considered infeasible. Existing strain-aware genome assemblers either do not scale well to larger genomes or more haplotypes [20], or use heuristics for path extraction instead of decomposing the flow minimally [21], [22].

Recent advancements into integer linear programming (ILP)-based flow decomposition have yielded a more performant framework for solving MFD in full [2], [23], [24]. Huizenga [1] has combined these results to build a new algorithm for strain-aware genome assembly, using the ILP formulation given by Dias et al. [2] to solve MFD with subpath constraints. Exploring the entire solution space brings a modest increase in assembly accuracy. However, this comes at the cost of a heavily increased runtime, to the point that it is computationally infeasible to use this method when around six or more haplotypes are present within the sample, often failing to converge even when provided tens of hours of compute time on high performance computing infrastructure [1].

Adding the subpath constraints greatly improves the convergence time of the ILP. On the other hand, if there are misassemblies among the contig set, overly stringent subpath constraints lead to an overestimation of the number of strains present in a sample [1], reducing performance. In this research, we formalize and compare various relaxations of the subpath constraints in the ILP to determine which formulation might be preferred based on the specifics of the data at hand. This yields the following research question:

What is the effect of different subpath constraint formulations on the performance of the ILP to solve MFD for viral genome assembly under varying levels of noise?

To answer this question, we formalize four parametrized subpath constraints formulations, three of which are relaxations of the typical formulation of subpath constraints [3], and implement these formulations in a contribution to the open source flow decomposition toolkit *flowpaths* [4]. We test against CVGs generated from real assemblies, and on artificially generated CVGs to isolate runtime effects and noise resilience.

We show that the three relaxations considered in this research affect the runtime negatively, and that the fourth formulation could help quicken convergence if further research is performed to eliminate an issue arising in the feasibility of decomposing some CVGs under this formulation.

2 Background

2.1 Contig variation graphs for strain-aware genome assembly

Variation graphs arise in a natural way when thinking about genomes that share some regions, but have significant differences as well. Variation graphs encode these genomes by assigning the nucleotide sequence of each such region to a node and adding a directed edge between two nodes when their sequences occur directly after each other in one of the genomes [18]. In this manner, the variations of a whole quasispecies can be encoded into a single graph. However, if we had the full haplotypes already, assembly would not be necessary, so we attempt to construct this data structure from the most complete genome sequences available at this point in assembly: the contigs.

Mathematically, we define a *contig variation graph* to be a tuple $(V, E, \mathcal{S}, a)$. Here (V, E) is a directed graph with each node $v \in V$ representing a nucleotide sequence $\mathrm{seq}(v) \in \{\mathrm{A}, \mathrm{C}, \mathrm{T}, \mathrm{G}\}^*$. We define in V unique source and sink nodes s and t such that $\deg^-(s) = \deg^+(t) = 0$. The set $\mathcal{S}$ of simple paths in V represents the contigs, and $a : V \to \mathbb{R}_+$ is a function assigning to each node an abundance value.

Construction of the CVG from a set of contigs is performed by aligning the contigs against each other then iteratively combining aligned sequences into a variation graph [25]. Then, the graph is simplified by compressing non-branching paths into single nodes [1], [20], [21]. Finally, s and t are introduced to V, where s is connected to all nodes $u \in V$ with $\deg^-(u) = 0$ and all nodes $v \in V$ with $\deg^+(v) = 0$ are connected to t. Now, all haplotype genomes that can be pieced together from the contigs are encoded as s–t paths in the CVG. These include many more genome assemblies than there are true haplotypes, as every variation in the CVG increases the amount of different s–t-paths exponentially.

The reads provide one more crucial bit of information which can allow us to reconstruct the true haplotypes from a CVG $(V, E, \mathcal{S}, a)$. By mapping the reads onto nodes $v \in V$ of which there is a sufficient overlap with $\mathrm{seq}(v)$, and counting how many reads were mapped to each node, we can calculate the approximate abundance of $\mathrm{seq}(v)$ within the sample and we assign this value to $a(v)$. We make the (sometimes problematic, see subsection 6.2) assumption that every haplotype has consistent coverage in the reads throughout the length of the genome. Then, the s–t-paths we use can be weighted by the abundance of their respective haplotype within the sample. If the weights of all s–t-paths in our set of haplotypes going through a node $v \in V$ are summed, we expect to find the total abundance of the node's sequence $\mathrm{seq}(v)$, given by $a(v)$. This way, we can quickly check if a set of genomes could generate the observed reads.

Decomposing the CVG into the right s–t-paths is the hard part of graph-based strain-aware genome assembly. Available full-genome assemblers using this approach include Virus-VG [20], VG-Flow [21], and VStrains [22]. Virus-VG enumerates all possible s–t-paths, which is accurate but not scalable beyond very small CVGs. VG-Flow solves a computationally easier problem on an auxiliary data structure to estimate abundances on the level of contigs, and builds genomes up from these contigs with their abundance. This is very fast, but assigning abundances to contigs loses information as abundance is not homogeneous throughout a contig (some sequences are shared with other contigs and have higher abundance). VStrains uses a heuristic algorithm to iteratively untangle the CVG and grow contigs out into full genomes, but also loses information in the process. We explore a way to decompose the CVG into s–t-paths without losing information in the next subsection.

It must be noted that the definition of a contig variation graph does not include the reverse complement of a node sequence into the node set, unlike the original definition of a (non-contig) variation graph [25]. That would be less useful to us, as our contigs are all forward-oriented after assembly. Most definitions of contig variation graphs allow for cycles [20], [21]. Cycles arise naturally from repetitive sequences in haplotype genomes. Notable examples of such repetitions are the long terminal repeats of HIV strains [26]. Dealing with cyclic CVGs is difficult, requiring specialized (and more complex) models [27] beyond the scope of this research. In the rest of this section we will therefore implicitly assume our CVGs to be acyclic.

2.2 Integer linear programming for minimum flow decomposition

Before we explain how to extract full genomes from the CVG in an optimal manner, we define a slightly richer structure for our CVG to formalize the decomposition explained in subsection 2.1. A *flow network* (V, E, f) is a DAG (V, E) with distinct source s and sink t nodes, combined with a flow function $f : E \to \mathbb{R}_+$ on the edges such that the sum of flow on incoming edges equals the sum of flow on outgoing edges on any node except s and t. This is a logical condition to impose, as a weighted s–t-path going through $v \in V$ must contribute the same weight to the incoming and outgoing edges of v. A *flow decomposition* of this flow network is then a set of s–t–paths $\mathcal{P} = (P_1, P_2, \dots)$ together with weights $\mathbf{w} = (w_1, w_2, \dots)$ such that for any edge $e \in E$, the sum of the weights of the paths going through e equals $f(e)$. These definitions are standard in literature on flow decompositions. Note however that a flow network is defined as having flow values on edges, while the CVG has abundance

values on nodes. We can easily expand the node-weighted CVG into an edge-weighted graph by substituting every node with two nodes with a connecting edge transporting the flow in between. The toolkit we use performs this action automatically.

We aim to find the best explanation for the sequencing data at hand, and following the principle of parsimony that is the decomposition with the least amount of paths. As such, a Minimum Flow Decomposition is defined as a flow decomposition $(\mathcal{P}, \mathbf{w})$ minimizing $|\mathcal{P}|$. For a general flow network, an MFD is not unique, but since the CVG is likely to have very specific flow values due to specific abundances in the reads, we assume only one MFD exists. If there are in fact multiple MFDs, then the CVG does not provide enough information to assemble the haplotype genomes with certainty. Finding an MFD is NP-hard, as is approximating it up to any arbitrary degree of precision [19]. This presents inherent scalability issues with solving for the MFD for genome assembly, and is why currently most graph-based full genome assemblers use heuristics for path extraction [21], [22]. However, Dias et al. [2] have developed an Integer Linear Programming-formulation for finding a k-flow decomposition ($|\mathcal{P}| = k$), which, when combined with a capable ILP solver, is typically faster than specialized algorithms. The runtime of this approach tends to increase exponentially for larger k, such that every increment in k tends to be more expensive than calculating the decomposition for all lower values of k summed [1]. Therefore simply iterating over k is satisfactory to solve MFD.

Before this ILP is formally expressed, there remains one more problem to deal with. CVGs are typically not flow networks, as noise and inconsistencies in the original reads and misassemblies in the contigs propagate up and result in imprecise abundance values on the nodes or topological errors. This can either be solved by correcting the abundance values to form a perfect flow beforehand [28], or by incorporating a penalty for inaccurate flows into the (otherwise empty) objective function of the ILP. This second option has also been explored in literature, with two formulations published. The first quantifies the error as the sum of differences (either absolute [4] or squared [29]) between the target edge flow values and the actual flow. The second defines the error as the sum of path errors, where the path error is the maximum difference between the weight of the path and its contribution to the edge flow value [30]. The path error minimization approach relies on a consistent error amplitude throughout the genomes, which unfiltered CVGs typically do not have due to regions with lower read coverage at the edges of the genome. Trial experiments were run using the first approach mentioned, where the flow is corrected beforehand. This did not result in easily decomposable graphs, which is further explained in section 6. Therefore we focus our research on the approach that minimizes the per-edge flow error, which was already proven to be effective on realistic CVGs by Huizenga [1]. The implementation we are using is part of the open source flow decomposition Python package *flowpaths* [4], where it is implemented in the method `kLeastAbsErrors`. For the full ILP, we refer to *flowpaths* [4] and Dias et al. [2]. As a short summary, we encode for all paths $i \in \{1, \ldots, k\}$ a weight w_i and a variable x_{uvi} to denote whether the edge $(u, v) \in E$ is used in path i. The variables are constrained to ensure each edge set is an s–t-path, and the sum of the absolute flow error on each edge is minimized. We will later use the variable x_{uvi} to enforce our subpath constraints.

When solving for the (perfect) MFD, we can simply iterate over k until we find a feasible solution. This is more complex for `kLeastAbsErrors`, as a feasible solution with a high error may not be preferred over a lower error solution using more paths. We can iterate over k until the error stabilizes, but this requires at least one more iteration than strictly necessary – which is rather expensive due to the exponential runtime scaling. However, it was found that the number of paths in the Minimum Path Cover (MPC) is a good predictor for k [1], and we can use this value to avoid iterating. MPC is a problem related to MFD that searches for a set of paths $\mathcal{P}$ that cover each edge in a DAG at least once, minimizing $|\mathcal{P}|$ and respecting subpath constraints if defined. Finding an MPC is solvable in polynomial time [31]. An ILP formulation for MPC and its implementation is given in *flowpaths* in the method `MinPathCover`, further streamlining the decomposition process.

2.3 Subpath constraints

Since the contigs are already known and assumed to be error-free contiguous subsequences of the full genomes, we can refine MFD by requiring the contig sequences to appear in the solution path. The efficacy of restricting MFD to use subpaths in the solution paths was known even before the ILP outlined in subsection 2.2 was formulated: the CVG together with contig subpaths encodes more information than the CVG itself, and assembly performance can be improved by including subpath constraints [3]. Determining in which manner to use the subpath information for the best performance is the object of this research. We define the set of subpaths $\mathcal{S}$ as a set of simple paths in a flow network (V, E, f). In our application, $\mathcal{S}$ will be given by the contigs in the CVG $(V, E, \mathcal{S}, a)$.

Typically, each subpath $S \in \mathcal{S}$ is required to appear in its entirety in at least one of the solution paths $P \in \mathcal{P}$ for the constraint to be satisfied. However, preliminary, non-systematic results from recent research [1] show that a relaxed formulation, inspired by the heuristic graph-based full genome assembler VG-Flow [21] can solve some cases which take too much compute time when using the standard subpath constraint formulations. This is why we investigate the subpath constraints further.

3 Methods

3.1 Subpath constraints formulations

As discussed in subsection 2.3, subpath constraints force specific sequences of nodes to appear in the solution path. The effects of the manner in which we enforce this are what we intend to explore in this research.

We name the formulation typically seen in the literature as *all-used* [2], [3]. In this formulation, every subpath must be *covered* by some solution path, *i.e.*, all edges of the subpath need to appear in at least one solution path together. We obtain one relaxation by reducing the strictness of this coverage:

we define *covered-p* to enforce that every subpath needs to be covered for a portion p. That is, a portion p of the edges of this subpath, weighted by the corresponding sequence lengths of the edges, must appear in at least one solution path together. Note that this need not be a contiguous sequence.

Then we define two closely related relaxations that allow the solver to skip some problematic subpaths. We take *p-used* to enforce that a portion p of all subpaths are covered in full by some solution path. We define *p-used-by-length* similarly, but now the portion p is weighted by the sum of the sequence lengths of the edges in a subpath.

A more complicated formulation is that of *only-subpaths-p*. Under this formulation, we require every edge in every solution path to appear in at least one subpath, where that subpath is covered by the solution path in question for a portion p, defined analogously to covered-p.

Finally, we identify *none* as a formulation in which the subpaths are disregarded entirely.

An overview of all the mathematical definitions and implementations of these subpath constraints formulations is given in table 5 in appendix A. Compared to all-used, the formulations p-used, p-used-by-length, and covered-p are relaxations. The formulation only-subpaths-p is neither strictly a relaxation nor a restriction. The formulation only-subpaths-p arises from a step in the pipeline of the strain-aware full genome assembler VG-flow [21], in which the flow is decomposed on a structure called a flow variation graph, effectively enforcing only-subpaths-1. It was then further shown to slightly outperform all-used in a small test on realistic CVGs [1].

In the case of small misassemblies, such as minor insertions, deletions or point errors, we expect covered-p to outperform other formulations. This formulation requires all subpaths to be covered, but only for a set portion of its edges. This allows the solver to effectively skip bad regions of the subpaths. In the case of larger errors, such as chimeric contigs, p-used and p-used-by-length are expected to perform better. These formulations allow us to disregard bad subpath constraints in their entirety. The formulation only-subpaths-p can possibly reduce the solution space while still dropping problematic subpaths and, in theory, quicken convergence. Whether the parameter p in this formulation is of any use is as of yet uncertain.

We implement all of these formulations in a contribution to *flowpaths* [4]. As of the defense of this thesis, the contribution is in a draft phase, as documentation and unit tests are still being developed.

3.2 Generating artificial CVGs with varying levels of noise

To isolate the effects of noise in the contigs on the performance of the decomposition, we require a dataset of similar CVGs with a known, increasing number of contig misassemblies. For this, we use a method for generating artificial CVGs from scratch, developed by Huizenga [1], and add our own method to induce misassemblies.

This procedure starts by sampling a genome of length N from $\{\mathrm{A}, \mathrm{C}, \mathrm{T}, \mathrm{G}\}^N$ randomly. Then we create a set of related haplotype genomes by inducing single-nucleotide polymorphism (SNP) mutations at a specified rate throughout the genome. This keeps the genome length invariant, and avoids having to perform alignment later on. Huizenga generates every haplotype as a mutated strain from the original genome, but we allow mutations on haplotypes to form new haplotypes, similar to how a quasispecies would evolve.

In the next step we build the haplotypes into a CVG. This is considerably easier than building a CVG from contigs, especially since our genomes are of the same length and alignment is not necessary. We just iterate through the genome per nucleotide, add a node for all different nucleotides at this index, and connect nodes that form a contiguous sequence in one of the haplotype genomes. Simultaneously, the genomes are sliced into slightly overlapping contigs of equal length, and these are assigned to the nodes of their sequence as well.

A notable difference here is that Huizenga makes contigs by taking the same slice (with the same indices) for all haplotype genomes, while we also develop a method that adds random noise as offsets to the genomes. This prevents the formation of regions where many multiples of contigs overlap, while the rest of the CVG is comparatively easy to solve. On the other hand, such regions can occur in real CVGs as well. We aim to test both approaches.

Finally, non-branching paths in the CVG are compacted into a single node while preserving subpaths. We add an extra step to filter out invalid contigs, such as single-node subpaths and subpaths that occur as proper subsets of another subpath.

We then induce misassemblies into the contig set by changing subpaths. Specifically, we introduce chimeric contigs, which are made up of sequences from different genomes and should therefore not appear in a proper assembly. The way we introduce these contigs is by randomly selecting a pair of overlapping contigs, identifying all regions in which they overlap, and then swapping the regions around these overlaps. The resulting contigs will still overlap in the same regions, but the regions in between these overlaps can now stem from different genomes.

4 Experimental setup

The performance of the following subpath constraint formulations is tested:

- all-used (baseline)
- 0.9-used
- covered-0.9
- only-subpaths-1
- none

The choice for this subset of the formulations we have defined stems from their role as representatives among the larger set. For example, if we find that 0.9-used performs well, we can further test 0.8-used, 0.95-used and 0.9-used-by-length. We do not change p in only-subpaths-1, as this formulation itself is already rather different from the baseline, so we want to test the unmodified formulation first.

Next, we specify the pipeline used to decompose CVGs under these constraints. To estimate k, MPC is used along with the particular formulation of subpath constraints tested. Next, `kLeastAbsErrors` (kLAE) is run on the CVG, specifying k

Dataset: realistic HIV CVGs				
number of strains k within sample	2	3	4	5
amount of acyclic CVGs	10	8	10	8

Table 1: The amounts of acyclic CVGs in the HIV dataset [32] for each k.

Dataset: Artificial CVGs				
number of strains k within sample	2	3	4	5
amount of CVGs	30	30	30	30

Table 2: The amounts of (acyclic) CVGs in the artificially generated dataset (see 3.2) for each k.

as estimated. Based on results from Huizenga [1] we set a time limit of 20 minutes for the MPC step and 60 minutes for kLAE. Especially the time limit for kLAE is short, and we do not expect all samples to decompose successfully. We keep track of the MPC result, runtimes, and result paths, as well as any errors if they appear.

We test on two datasets, the first to evaluate performance in a realistic setting. For this we use CVGs assembled by Huizenga [1] from simulated reads of the HIV genome that underwent assembly with SAVAGE [17]. LTRs [26] were removed to avoid creating cycles.

Ten CVGs are provided for all true haplotype counts $k \in \{2, 3, 4, 5\}$. Still, some of these CVGs do contain cycles. As we specified in subsection 2.1, we assume that CVGs are acyclic, and therefore we simply discard these cyclic CVGs from the dataset. The number of samples left for each number of strains is shown in table 1.

The second dataset consists of CVGs artificially generated from the procedure outlined in subsection 3.2. Genomes are sampled at length 2000 bp, the SNP mutation rate is set to 0.005, and the contig length is 100 bp, of which 20 bp overlaps with the next contig. We generate 5 CVGs for every true haplotype count $k \in \{2, 3, 4, 5\}$, for static contig slicing and for our randomized contig slicing, and for chimeric contigs induced at noise levels 0.0, 0.1, and 0.2. In total, this makes for 120 CVGs. Again, the number of CVGs for each k is shown in table 2.

Each of the samples is run with every subpath constraints formulation individually on the high performance computing (HPC) system DelftBlue [33]. We give every sample 16 CPU cores and 64 GB RAM. We analyze the results of these runs in the next section.

5 Results

The results are grouped according to the dataset they are generated from.

5.1 HIV dataset

We first analyze which samples managed to finish within the 1-hour time limit imposed on the ILP solver. There are ten identically sampled CVGs for $k \in \{2, 3, 4, 5\}$ true haplotypes, and every CVG has been run under the five subpath constraints formulations expounded in section 4, so we group

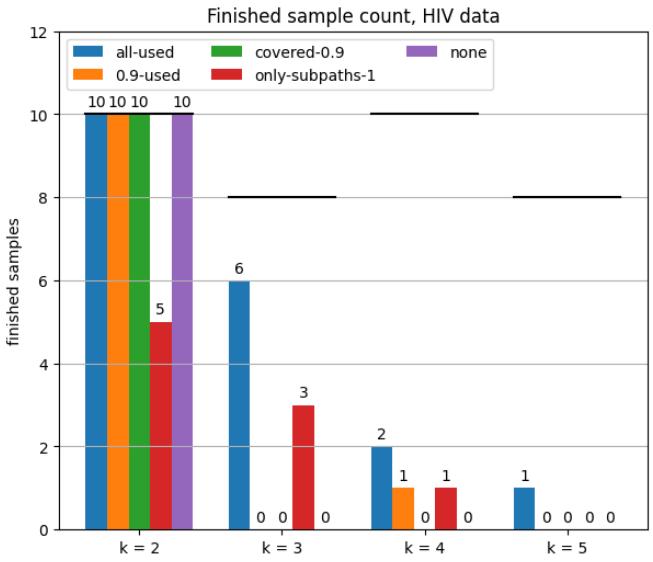


Figure 1: The amount of finished samples in the HIV dataset, grouped by true amount of strains k and subpath constraints formulation used. The amount of total samples for a particular k is indicated by the black line.

the results on k and the formulation used. The results are presented in figure 1.

It is clear that all formulations managed to decompose fewer CVGs as the amount of true haplotypes k increases. From the formulations tested, the most samples completed under the standard subpath constraints formulation all-used for every k. In fact, only all-used and only-subpaths-1 can consistently complete some samples for $k \geq 3$, with covered-0.9, 0.9-used and none failing for all but one CVG in this category. Lastly we find that every formulation except only-subpaths-1 can decompose all $k = 2$ graphs. This contrasts the results for $k \geq 3$, where only-subpaths-1 beats the three relaxations.

High runtimes can be caused by overestimation of k during the MPC step [1]. To investigate if this plays a role, we present the MPC estimates for k in table 3. We can see that covered-0.9 and none achieve perfect results here by estimating the true k on every sample. On the other hand, only-subpaths-1 completes only 9 out of 36 path covers. This warrants further investigation, as unlike MFD or kLAE, MPC is solvable in polynomial time [31].

The outcomes of all runs with only-subpaths-1 are outlined in figure 2. We have found that a majority of CVGs are infeasible to cover and decompose for any k. Our results leave it ambiguous whether the MPC TIMEOUTs are due to infeasible graphs, or simply because solving for an MPC takes longer than the allotted 20-minute time limit. Note that unlike the other formulations, which result almost exclusively in successes or decomposition timeouts, only-subpaths-1 never encountered a decomposition timeout in this dataset.

Finally we can analyze the runtimes of all samples that managed to complete in figure 3. Take note of the logarithmic scale for runtime: the lower quarter of the graph represents runtimes of less than one second. It appears that the relax-

constraints	< -2	-1	0	1	> 2	total
all-used	0	0	16	16	2	34
0.9-used	0	0	34	0	0	34
covered-0.9	0	0	36	0	0	36
only-subpaths-1	0	0	9	0	0	9
none	0	0	36	0	0	36

Table 3: MPC estimates for k compared to true k in the HIV dataset, grouped by subpath constraints formulation used. In this table, > 2 means the MPC estimate is two or more above the true k. Note that there are 36 samples in total; formulations that score less than 36 on the total MPC estimates ran into timeouts during the MPC stage of some samples.

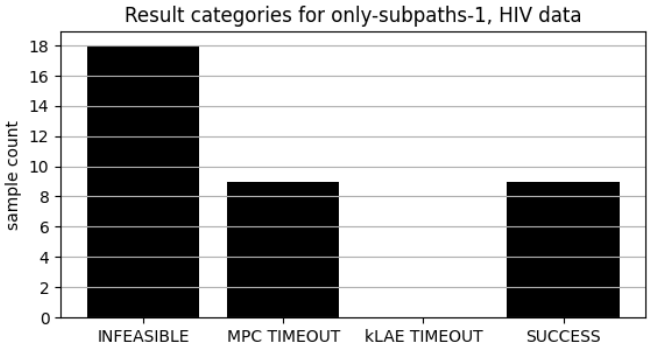


Figure 2: Categorized results for only-subpaths-1 in the HIV dataset. Note that an MPC TIMEOUT may also be indicative of an infeasible CVG, so in actuality the number of infeasible CVGs may be higher than indicated solely by INFEASIBLE. Note also that no sample encountered the time limit on the `kLeastAbsErrors` decomposition.

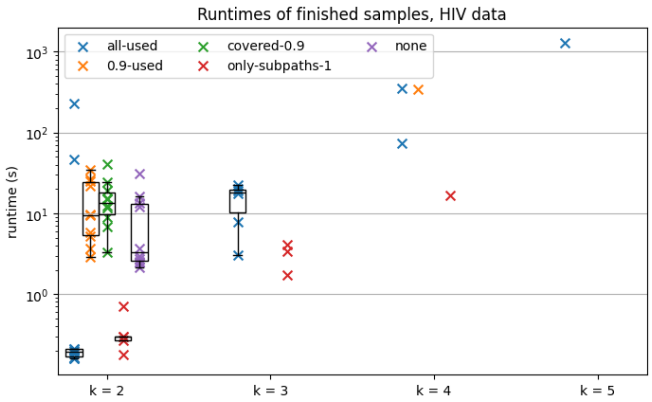


Figure 3: Runtime of the `kLeastAbsErrors` decomposition of finished samples in the HIV dataset, logarithmically scaled. The runtimes are grouped by true k and subpath constraints formulation used. We include quartile boxplots for the categories where 5 or more samples completed.

ations covered-0.9, 0.9-used and none have generally greater runtimes than all-used, although for $k = 2$ there are two outliers with all-used that take longer to decompose than all results from the three relaxations mentioned, one of these outliers by a factor of around 10. We discuss only-subpaths-1 separately. As seen before, this formulation often encounters infeasibility in the MPC step (which implies decomposing is also impossible). However, when decomposition is possible, it scores close to all-used for $k = 2$ and performs even better than all-used on $k \geq 3$, by a factor of around 8 for $k = 4$.

5.2 Artificial CVG dataset

We have more groupings to consider for the artificial CVG dataset. The samples can be split into two groups based on contig slicing methodology used, into three groups based on the rate at which contig misassemblies were introduced, and four groups based on the number of true haplotypes $k \in \{2, 3, 4, 5\}$, as before. We first consider the contig slicing methodology, since aside from the subpaths the two categories consist of identical CVGs, and for later results we will choose to either present one or the other.

In figure 4 the number of finished samples under the two contig slicing methods are compared. We can see that for all formulations tested except only-subpaths-1, the slicing method does not affect how many samples complete. For only-subpaths-1 however, there is a stark difference, with more CVGs being able to be decomposed when they are generated with static contig slicing.

Next we look at the MPC estimates for these CVGs in table 4. The results here are divided by the induced misassembly rate as well as the subpath constraints formulation used to see if noise in the contigs has an effect on MPC results. It appears there is a subtle correlation, with a higher misassembly rate resulting in some overestimations of k with all-used and also once for 0.9-used. However, in general all formulations underestimate k much more frequently than they overestimate, especially for covered-0.9 and none. For these formulations k is underestimated in a majority of the samples. The con-

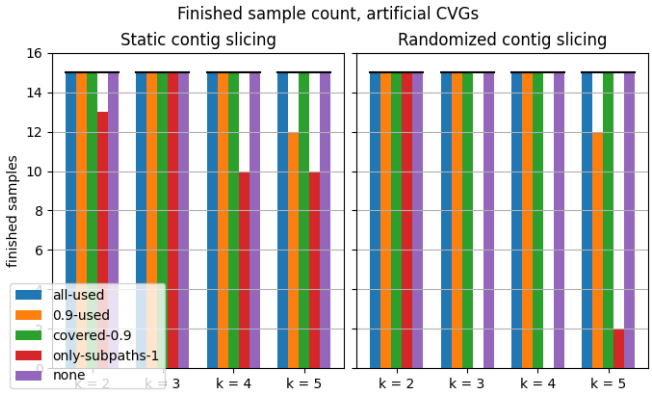


Figure 4: The number of finished samples in the artificial dataset, grouped by true number of strains k, subpath constraints formulation used and separately given for static contig slicing and randomized contig slicing. The number of total samples for a particular k is indicated by the black line.

constraints	< -2	-1	0	1	> 2	total
Contig misassembly rate: 0.0						
all-used	0	11	29	0	0	40
0.9-used	7	13	20	0	0	40
covered-0.9	20	8	12	0	0	40
only-subpaths-1	7	8	11	0	0	26
none	20	8	12	0	0	40
Contig misassembly rate: 0.1						
all-used	0	9	28	2	1	40
0.9-used	7	12	21	0	0	40
covered-0.9	20	8	12	0	0	40
only-subpaths-1	4	5	12	0	0	21
none	20	8	12	0	0	40
Contig misassembly rate: 0.2						
all-used	0	10	26	3	1	40
0.9-used	7	13	19	1	0	40
covered-0.9	20	8	12	0	0	40
only-subpaths-1	2	2	14	0	0	18
none	20	8	12	0	0	40

Table 4: MPC estimates for k compared to true k in the artificial dataset, grouped by induced contig misassembly rate and subpath constraints formulation used. In this table, > 2 means the MPC estimate is two or more above the true k. Note that only-subpaths-1 is not feasible on every CVG, and therefore not all samples have completed successfully on this formulation.

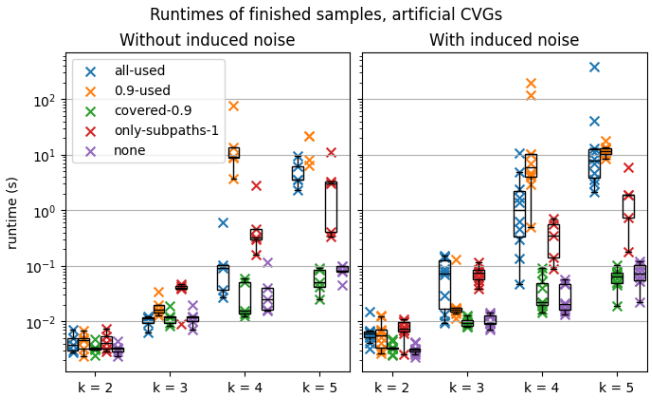


Figure 5: Runtime of the `kLeastAbsErrors` decomposition of finished samples in the artificial dataset, with quartile boxplots, logarithmically scaled. We include only the samples generated with static contig slicing. The runtimes are grouped by true k, subpath constraints formulation used, and whether artificially induced contig misassemblies are present.

tig misassembly rate has no effect on these formulations either, which is trivial for none but notable for covered-0.9. For only-subpaths-1, the misassembly rate seems to correlate positively with the number of infeasible samples.

Then finally the runtimes of the artificial CVGs are considered. To avoid duplication, we choose to only present the results with static contig slicing, as these see more completed samples for only-subpaths-1. We do note that including also the samples with randomized contig slicing does not paint a different picture, except that a lower fraction of only-subpaths-1 decompositions are completed. The CVGs are again divided into samples with induced misassemblies and those without. We merge the categories for misassembly rate 0.1 and 0.2, as the difference seems minor in table 4.

The results are presented in figure 5. Here we see a clear correlation between the true k and the runtime, although we note there is much more variance at $k \geq 4$. Again, consider the logarithmic scale, which makes the runtimes for higher k seem closer together than they truly are. At $k = 4$, the difference between the quickest all-used decomposition and the slowest is around 500-fold. Contrary to the HIV dataset, here the relaxations covered-0.9 and none consistently perform the fastest decompositions. The formulation 0.9-used has runtimes overlapping with the higher end of all-used, but notably on $k \geq 4$ it has runtimes orders of magnitude greater than the other formulations. The formulation only-subpaths-1 does not seem comparable to all-used on the samples without contig noise, but has consistently lower runtimes on the samples with misassemblies, at least on the samples that are feasible.

6 Discussion

6.1 Implications and suggestions for further research

Now we return to our research question and evaluate whether any of the formulations we tested are preferable to the standard all-used. We first formulate some intermediate conclu-

sions. It appears that the MPC is an excellent predictor of the true k on the HIV data. This is consistent with Huizenga's results [1]. However, its predictive power is much worse on the artificial CVGs. This is not necessarily inconsistent with Huizenga's results on the MPC either, as there it is specifically concluded that the MPC is a good predictor for well-formed graphs, and our artificial CVGs are arguably not well-formed (see subsection 6.2). We conclude that for such particular CVGs another measure must be used to estimate k, or we can simply iterate until the error stabilizes.

Second, it seems that the formulations 0.9-used, covered-0.9 and none are unsuitable for use on realistic CVGs. Their performance on the HIV dataset for $k \geq 3$ is simply insufficient for this purpose, as almost no samples converge within the time limit.

We contrast this with the performance of covered-0.9 and none on the artificial CVGs, where they seem to perform much better than every other formulation. However, those results cannot be interpreted without also considering the MPC results. The ubiquitous underestimations of the true k for covered-0.9 and none imply that the assembly quality must be negatively affected. Of course, it is much easier to decompose the CVG if we use fewer paths than there are haplotypes, but then the assembly cannot achieve a good result. That is why we still consider the formulation all-used to be superior in this situation. Since again 0.9-used generally has greater runtimes on the artificial CVGs, we also consider this formulation to be less useful than all-used.

That leaves the formulation only-subpaths-1 to be examined. In its current state, the feasibility issues observed in the results make this formulation less useful than all-used. However, we observe that for the samples where MPC is feasible, the runtimes are generally better than all-used can achieve. We also note that Huizenga makes no mention of these feasibility issues in her research [1], where only-subpaths-1 is evaluated on different CVGs under the name VG-flow constraint, after their original inspiration. This raises the question if the feasibility problem is an artifact of our implementation, but we have verified its correctness manually. Also, mathematically, feasibility is not guaranteed for only-subpaths-p, as an edge (u, v) may be assigned to only one subpath, which appears on only one branch before or after (u, v). In this case, any other branch cannot be used, as that would imply using (u, v) in a solution path without covering its only subpath.

We see this as an excellent starting point for future research. If there exists a formulation that can bring the runtime benefit of only-subpaths-1 without bringing about infeasibility, then that can provide a major boost in runtime for the decomposition. We suggest looking into relaxations such as $p < 1$, or preprocessing steps such as untangling problematic edges into separate branch paths similar to VStrains [22]. The latter could be considered a heuristic, but by extending the ILP to allow summing over multiple edges when calculating a single flow error, we would still explore the full solution space.

6.2 Limitations of this research

We mention a number of important limitations one should be aware of when drawing conclusions from the results presented in this paper.

Evaluating accuracy

Of course, runtimes alone do not tell the full story when it comes to what makes a set of subpath constraints usable in practice. The effect different constraints have on the accuracy of the reconstruction is just as important. Especially on the perfect graphs, where the results show that the number of true haplotypes is often underestimated significantly by MPC, we expect there to be serious problems with the assembly quality. The stringent time constraints on this project have forced us to abandon evaluating accuracy metrics on the reconstructed haplotypes, but we recommend that anyone doing further research into this subject add an assembly quality evaluation step to their pipeline to get a more complete set of results. A simple performance evaluation metric we implemented is described in section 6.3, but if time allows it we recommend the industry standard for strain-aware assembly quality evaluation, MetaQUAST [34], be used instead.

Facile topology of artificial CVGs

As specified in section 3, we base our methodology for generating artificial CVGs on the approach presented by Huizenga [1]. While our additions to the generation pipeline remedy some problems, the overall topology of the resulting CVGs remains similar to the perfectly decomposable graphs used in the original research. This structure is rather simple, and consists almost entirely of a single path splitting up and then immediately merging back together. Initial trial runs with the generation script have shown that this approach is capable of generating more intricate topologies as well, but this rapidly increases the decomposition difficulty to the point of infeasibility with the decomposition pipeline explained in section 4. If this is still desired, increase the SNP mutation rate in the data generation script, and limit the genome length to prevent overly long CVGs. The reason we choose such a simple structure is that complexity can then be induced by the subpaths; it allows us to isolate the effect of only the subpath constraints on otherwise easily decomposable CVGs.

Idealized coverage of simulated reads

The simulated HIV reads from which the contigs and CVG for the realistic experiments are built, feature consistent coverage throughout the genome. This is often not the case in real sequenced reads, where coverage can be highly dependent on the location within the genome. This raises some concerns about how well our conclusions will generalize to assemblies from real reads.

Technical limits

Due to limits on our access to DelftBlue, we cannot ensure that all experiments are run on the same exact hardware, only that the same number of cores and amount of memory are allocated. Some intermediate results from peers do seem to indicate that there is indeed some variance in performance across different nodes.

6.3 Partial work

We have researched or implemented a number of methods that did not fit into the overall structure of the experiments or

had to be discontinued for other reasons, but could still prove valuable to future research. These methods are outlined in this subsection.

Experiments with `MinErrorFlow`

The package *flowpaths* [4] implements a method to correct an arbitrary graph to a perfect flow network, that is to say, to change edge weights in a minimal manner to ensure incoming and outgoing flow is equal on every node [28]. This approach can then theoretically be followed up by the perfect MFD instead of `kLeastAbsErrors`. We have experimented with this approach on a number of 2-strain HIV samples from the dataset. The edge flow values were indeed corrected to form a perfect flow, but these values were not the same throughout the CVG. In this case, the CVG is not easily decomposable with MFD and the number of paths needed approaches the number of different flow values on edges in the graph. There exists a parameter `few_flow_values_epsilon` which can reduce the amount of distinct edge flow values generated. However, this is heuristic and there is still no guarantee that the corrected network will be nicely decomposable. Additionally, we see no clear method to estimate this parameter based on the characteristics of the graph. Therefore this approach could not be used in this research.

SANTA-SIM to simulate haplotype mutations

To overcome the issues with the artificial CVGs explained in subsection 6.2, we have applied a more realistic method to simulate haplotype evolution. By setting up a small-scale simulation experiment with the forward-time quasispecies simulator SANTA-SIM [35] we can extract haplotypes with realistic single nucleotide and insertion/deletion (*indel*)mutations. This approach was discontinued due to time limitations on the project and the need to use an aligner for the CVG construction if indels are present in the haplotypes. The code to sample haplotypes with SANTA-SIM is fully implemented and available.

Inducing noise on pre-assembled CVGs

This research was originally planned to induce contig misassemblies on already assembled CVGs, such as the HIV dataset, instead of creating new artificial CVGs. Misassemblies are induced in the contig set, and the CVG is reconstructed from the now noisier contigs, including a new read mapping. Besides the chimeric contig misassemblies explained in section 3, two more mutation categories are implemented: splitting contigs into two parts while leaving a gap, and single point mutations. The intent of this approach was to evaluate the effect of misassemblies in a realistic setting, and experiment with the effect of specific subpath constraints formulation on a single category of misassembly, such as missing coverage or chimerics. This was discontinued due to the limited performance on the HIV dataset without noise added, and also due to time limitations. Again, the code to induce such misassemblies is implemented and explained in the publicly available codebase.

A(n overly) simplistic accuracy metric

We have implemented a simple metric to evaluate the accuracy of a full genome assembly against the ground truth, by greedily matching the highest weight haplotypes from both sets, calculating the edit distance for each pair divided by the ground truth length and calculating the mean of these distances over all ground truth strains. This metric is inferior to a proper evaluation suite like MetaQUAST [34] and is only included here for the sake of completeness. Due to time limitations the metric has not seen any use during the project.

7 Conclusion

Subpath constraints improve convergence time in the ILP formulation for MFD [2] and help strain-aware viral genome assemblers build a more accurate set of haplotype genomes from contig variation graphs [1], [3]. We have mathematically formalized and implemented four alternative formulations for such subpath constraints in a contribution to the open source flow decomposition toolkit *flowpaths* [4]. We evaluate the effects of these formulations on the runtime by benchmarking the decomposition process on two datasets. The first of these consists of realistic contig variation graphs from prior research [1]. The second dataset we have generated ourselves, consisting of contig variation graphs with intentionally misassembled contigs to evaluate performance under noisy conditions.

We conclude that three of the four formulations tested affect the runtime negatively compared to the baseline all-used, and suggest against further usage of these relaxations. The last formulation, only-subpaths-p, is not usable in its current form due to issues with infeasibility, but does show potential. The CVGs that are feasible under only-subpaths-p converge quicker than under any other formulation. Therefore we recommend further research be done with the object of resolving the feasibility issues we encountered.

8 Responsible research

8.1 Reproducibility

To ensure that any interested party can replicate the results achieved in this project, the source code for generating data, running the experiments, and for data analysis is publicly available at https://github.com/devedude/ilpmfd-subpaths. The data generation methods make frequent use of a source of randomness, so it is ensured that all random draws can be seeded by optional parameters. The code consists of a collection of Python scripts, some of which can be invoked directly from the command line and some of which are designed to be invoked from caller scripts, also included. Additionally, there are README files in each section explaining what the scripts do and how to run and configure them. Some scripts are intended to have access to high performance computing resources. This is reflected in the README, and Slurm[1] batch job files are available for these tasks. If your HPC system uses another job manager, porting should be fairly straightforward: the batch jobs just call a Python script. The README files are mainly targeted at those who wish to replicate the pipeline presented here. For when further understanding of the code is helpful or the pipeline needs to be modified, multiple measures are taken to

[1] Slurm is a job manager for HPC clusters.

ensure that the code is readable and understandable. Many scripts include extensive commenting and docstrings. Unfortunately, due to time constraints in the project,, no automated testing suite has been developed; we recommend anyone extending the codebase to perform their own tests.

The HIV CVGs used for subsection 5.1 are the same as those Huizenga used [1] and shared [32] and may be downloaded from there. Due to a technical issue the reads available at Huizenga's Zenodo repository are corrupted (as of v1), but these are not needed for the results presented in this paper. The ground truth haplotypes are also available in the repository for those interested in evaluating assembly quality.

8.2 Ethical considerations

Research using high-performance computing resources is inherently costly, both financially for the TU Delft and environmentally due to the electricity demands. Part of researching responsibly is to consider such costs and weigh them against the potential benefits of this research. We believe that the ultimate goal of improving treatment and medical understanding is worth these costs. As another argument, since one of the primary goals of this research is to further optimize an algorithm, any speedup achieved here can apply to all future implementations to hopefully save much more compute time than this project has used. Still, where possible, we have made sure to save computational resources. A notable example is in contig assembly: SAVAGE uses a lot of processing time to build contigs from reads, but as the contigs were still available in the form of subpaths in the assembled CVG files in Huizenga's dataset, these could be reused. This likely saved multiple days of compute time.

9 Declaration on the use of AI

We have occasionally used an AI tool to identify spelling and grammar mistakes during the writing of this document but aside from that no AI tools were used to facilitate this research in any manner.

10 Acknowledgements

I would like to thank Jasmijn Baaijens, my supervisor, for her continued support throughout the project and her technical suggestions during the research. I would also like to extend my gratitude to Alexandru Tomescu for his insight and help with developing the contribution to *flowpaths*. And finally I greatly thank my family and my girlfriend for supporting me and keeping me sane throughout this research project, I could not have completed it without them.

References

[1] R. R. Huizenga, "Minimum Flow Decomposition for Haplotype-Aware Genome Assembly," MSc thesis, Delft University of Technology, 2025. [Online]. Available: https://resolver.tudelft.nl/uuid:fe6b1f15-e029-48ae-9080-8cef87c771c0.

[2] F. H. C. Dias, L. Williams, B. Mumey, and A. I. Tomescu, "Fast, Flexible, and Exact Minimum Flow Decompositions via ILP," in *Research in Computational Molecular Biology*, 2022. DOI: 10.1007/978-3-031-04749-7_14.

[3] L. Williams, A. I. Tomescu, and B. Mumey, "Flow Decomposition With Subpath Constraints," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 1, 2023. DOI: 10.1109/TCBB.2022.3147697.

[4] A. Tomescu, *flowpaths Python Package*. [Online]. Available: https://algbio.github.io/flowpaths.

[5] S. Duffy, "Why are RNA virus mutation rates so damn high?" *PLOS Biology*, vol. 16, no. 8, 2018. DOI: 10.1371/journal.pbio.3000003.

[6] J. M. Cuevas, R. Geller, R. Garijo, J. López-Aldeguer, and R. Sanjuán, "Extremely High Mutation Rate of HIV-1 In Vivo," *PLOS Biology*, vol. 13, no. 9, 2015. DOI: 10.1371/journal.pbio.1002251.

[7] R. Geller et al., "Highly heterogeneous mutation rates in the hepatitis C virus genome," *Nature Microbiology*, vol. 1, no. 7, 2016. DOI: 10.1038/nmicrobiol.2016.45.

[8] Y. Liu et al., "The Genetic Diversity of HIV-1 Quasispecies Within Primary Infected Individuals," *AIDS Research and Human Retroviruses*, vol. 36, no. 5, 2020. DOI: 10.1089/AID.2019.0242.

[9] S. L. Fishman and A. D. Branch, "The Quasispecies Nature and Biological Implications of the Hepatitis C Virus," *Infection, Genetics and Evolution*, vol. 9, no. 6, 2009. DOI: 10.1016/j.meegid.2009.07.011.

[10] E. Domingo, "Quasispecies and the implications for virus persistence and escape," *Clinical and Diagnostic Virology*, vol. 10, no. 2, 1999. DOI: 10.1016/S0928-0197(98)00032-4.

[11] B. L. Guen, G. Squadrito, B. Nalpas, P. Berthelot, S. Pol, and C. Brechot, "Hepatitis C virus genome complexity correlates with response to interferon therapy: A study in French patients with chronic hepatitis C," *Hepatology*, vol. 25, no. 5, 1997. DOI: 10.1002/hep.510250531.

[12] E. Domingo and C. Perales, "Viral quasispecies," *PLOS Genetics*, vol. 15, no. 10, 2019. DOI: 10.1371/journal.pgen.1008271.

[13] T. Xiao and W. Zhou, "The third generation sequencing: The advanced approach to genetic diseases," *Translational Pediatrics*, vol. 9, no. 2, 2020. DOI: 10.21037/tp.2020.03.06.

[14] A. D. Radford, D. Chapman, L. Dixon, J. Chantrey, A. C. Darby, and N. Hall, "Application of next-generation sequencing technologies in virology," *Journal of General Virology*, vol. 93, no. 9, 2012. DOI: 10.1099/vir.0.043182-0.

[15] A. Töpfer, T. Marschall, R. A. Bull, F. Luciani, A. Schönhuth, and N. Beerenwinkel, "Viral Quasispecies Assembly via Maximal Clique Enumeration," *PLOS Computational Biology*, vol. 10, no. 3, 2014. DOI: 10.1371/journal.pcbi.1003515.

[16] J. Chen, Y. Zhao, and Y. Sun, "*De novo* haplotype reconstruction in viral quasispecies using paired-end read guided path finding," *Bioinformatics*, vol. 34, no. 17, 2018. DOI: 10.1093/bioinformatics/bty202.

[17] J. A. Baaijens, A. Z. E. Aabidine, E. Rivals, and A. Schönhuth, "De novo assembly of viral quasispecies using overlap graphs," *Genome Research*, vol. 27, no. 5, 2017. DOI: 10.1101/gr.215038.116.

[18] B. Paten, A. M. Novak, J. M. Eizenga, and E. Garrison, "Genome graphs and the evolution of genome inference," *Genome Research*, vol. 27, no. 5, 2017. DOI: 10.1101/gr.214155.116.

[19] 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*, vol. 185, no. 3, 2008. DOI: 10.1016/j.ejor.2006.05.043.

[20] J. A. Baaijens, B. V. der Roest, J. Köster, L. Stougie, and A. Schönhuth, "Full-length *de novo* viral quasispecies assembly through variation graph construction," *Bioinformatics*, vol. 35, no. 24, 2019. DOI: 10.1093/bioinformatics/btz443.

[21] J. A. Baaijens, L. Stougie, and A. Schönhuth, "Strain-Aware Assembly of Genomes from Mixed Samples Using Flow Variation Graphs," in *Research in Computational Molecular Biology*, 2020. DOI: 10.1007/978-3-030-45257-5_14.

[22] R. Luo and Y. Lin, "VStrains: De Novo Reconstruction of Viral Strains via Iterative Path Extraction from Assembly Graphs," in *Research in Computational Molecular Biology*, 2023. DOI: 10.1007/978-3-031-29119-7_1.

[23] A. Grigorjew, F. H. C. Dias, A. Cracco, R. Rizzi, and A. I. Tomescu, "Accelerating ILP Solvers for Minimum Flow Decompositions Through Search Space and Dimensionality Reductions," in *International Symposium on Experimental Algorithms*, 2024. DOI: 10.4230/LIPIcs.SEA.2024.14.

[24] F. Sena, R. Rizzi, and A. I. Tomescu, "Safe Sequences via Dominators in DAGs for Path-Covering Problems," in *European Symposium on Algorithms*, 2025. DOI: 10.4230/LIPIcs.ESA.2025.55.

[25] E. Garrison et al., "Variation graph toolkit improves read mapping by representing genetic variation in the reference," *Nature Biotechnology*, vol. 36, 2018. DOI: 10.1038/nbt.4227.

[26] F. Krebs, T. Hogan, S. Quiterio, S. Gartner, and B. Wigdahl, "Lentiviral LTR-directed expression, sequence variation, and disease pathogenesis," *HIV Sequence Compendium*, 2001. [Online]. Available: https://www.hiv.lanl.gov/content/sequence/HIV/COMPENDIUM/2001/partI/Wigdahl.pdf.

[27] F. H. C. Dias, L. Williams, B. Mumey, and A. I. Tomescu, "Minimum flow decomposition in graphs with cycles using integer linear programming," *Journal of Global Optimization*, vol. 93, 2025. DOI: 10.1007/s10898-025-01556-8.

[28] A. I. Tomescu, A. Kuosmanen, R. Rizzi, and V. Mäkinen, "A novel min-cost flow method for estimating transcript expression with RNA-Seq," in *BMC Bioinformatics*, 2013. DOI: 10.1186/1471-2105-14-S5-S15.

[29] W. Li, J. Feng, and T. Jiang, "IsoLasso: A LASSO Regression Approach to RNA-Seq Based Transcriptome Assembly," *Journal of Computational Biology*, vol. 18, no. 11, 2011. DOI: 10.1089/cmb.2011.0171.

[30] F. H. C. Dias and A. I. Tomescu, "Accurate Flow Decomposition via Robust Integer Linear Programming," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 21, no. 6, 2024. DOI: 10.1109/TCBB.2024.3433523.

[31] D. R. Fulkerson, "Note on Dilworth's decomposition theorem for partially ordered sets," in *Proceedings of the American Mathematical Society*, 1956. DOI: 10.1090/S0002-9939-1956-0078334-6.

[32] R. Huizenga, *Dataset minimum flow decomposition for haplotype-aware genome assembly*, Zenodo, 2025. DOI: 10.5281/zenodo.16910556. [Online]. Available: https://doi.org/10.5281/zenodo.16910556.

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

[34] A. Mikheenko, V. Saveliev, and A. Gurevich, "Metaquast: Evaluation of metagenome assemblies," *Bioinformatics*, vol. 32, no. 7, 2016. DOI: 10.1093/bioinformatics/btv697.

[35] A. Jariani et al., "SANTA-SIM: simulating viral sequence evolution dynamics under selection and recombination," *Virus Evolution*, vol. 5, no. 1, 2019. DOI: 10.1093/ve/vez003.

A Mathematical definitions and implementations of subpath constraints formulations

Identifier	Mathematical definition	ILP formulation
all-used	For all $S \in \mathcal{S}$ we have $S \subset P$ for some $P \in \mathcal{P}$.	$r_{ij} \in \{0,1\}$; $\sum_{(u,v)\in S_j} x_{uvi} \geq \lvert S_j \rvert r_{ij}$; $\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq 1$
p-used	There exists an $\mathcal{S}' \subset \mathcal{S}$ such that for all $S \in \mathcal{S}'$ we have $S \subset P$ for some $P \in \mathcal{P}$, and $\lvert\mathcal{S}'\rvert \,/\, \lvert\mathcal{S}\rvert \geq p$.	$r_{ij} \in \{0,1\}$; $r_j \in \{0,1\}$; $\sum_{(u,v)\in S_j} x_{uvi} \geq \lvert S_j \rvert r_{ij}$; $\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq r_j$; $\sum_{S_j\in\mathcal{S}} r_j \geq p\lvert\mathcal{S}\rvert$
p-used-by-length	There exists an $\mathcal{S}' \subset \mathcal{S}$ such that for all $S \in \mathcal{S}'$ we have $S \subset P$ for some $P \in \mathcal{P}$, and $\sum_{S\in\mathcal{S}'} \ell(S) \,/\, \sum_{S\in\mathcal{S}} \ell(S) \geq p$.	$r_{ij} \in \{0,1\}$; $r_j \in \{0,1\}$; $\sum_{(u,v)\in S_j} x_{uvi} \geq \lvert S_j \rvert r_{ij}$; $\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq r_j$; $\sum_{S_j\in\mathcal{S}} r_j\, \ell(S_j) \geq p \sum_{S_j\in\mathcal{S}} \ell(S_j)$
covered-p	For all $S \in \mathcal{S}$, there exists a $P \in \mathcal{P}$ such that $\ell(S \cap P) \,/\, \ell(S) \geq p$.	$r_{ij} \in \{0,1\}$; $\sum_{(u,v)\in S_j} x_{uvi}\, \ell(u,v) \geq r_{ij}\, p \sum_{(u,v)\in S_j} \ell(u,v)$; $\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq 1$
only-subpaths-p	For all $P \in \mathcal{P}$, there exists an $\mathcal{S}'_P \subset \mathcal{S}$ such that $(\cup P - \{s,t\}) \subset \cup\mathcal{S}'_P$ and for each $S \in \mathcal{S}'_P$ we have $\ell(S \cap P) \,/\, \ell(S) \geq p$.	$r_{ij} \in \{0,1\}$; $\sum_{(u,v)\in S_j} x_{uvi}\, \ell(u,v) \geq r_{ij}\, p \sum_{(u,v)\in S_j} \ell(u,v)$; $\sum_{S_j\in\mathcal{S}\mid(u,v)\in S_j} r_{ij} \geq x_{uvi}$
none	–	–

Table 5: An overview of the families of subpath constraints tested in this research, each with an explanation and the ILP variables and constraints required for implementation. In the above, we define the length of a set of edges $\ell(P)$ as the sum of the length of its elements. Additionally, we implicitly assume all indices in the ILP formulations are summed over unless defined otherwise. Here i is a solution path index, u and v are vertices, and j is a subpath index.