# Safety and Completeness in Flow Decompositions for RNA Assembly *

Shahbaz Khan[1,2][0000−0001−9352−0088], Milla Kortelainen[2][0000−0003−1590−0987], Manuel Cáceres[2][0000−0003−0235−6951], Lucia Williams[3][0000−0003−3785−0247], and Alexandru I. Tomescu[2][0000−0002−5747−8350]

[1] Department of Computer Science and Engineering, IIT Roorkee, India
shahbaz.khan@cs.iitr.ac.in
[2] Department of Computer Science, University of Helsinki, Finland
{shahbaz.khan,milla.kortelainen,manuel.caceresreyes,alexandru.tomescu}@helsinki.fi
[3] School of Computing, Montana State University, USA
luciawilliams@montana.edu

**Abstract.** Flow decomposition has numerous applications, ranging from networking to bioinformatics. Some applications require any valid decomposition that optimizes some property as number of paths, robustness, or path lengths. Many bioinformatic applications require the specific decomposition which relates to the underlying data that generated the flow. Thus, no optimization criteria guarantees to identify the correct decomposition for real inputs. We propose to instead report the *safe* paths, which are subpaths of at least one path in every flow decomposition.

Ma et al. [WABI 2020] addressed the existence of multiple optimal solutions in a probabilistic framework, which is referred to as *non-identifiability*. Later, they gave a quadratic-time algorithm [RECOMB 2021] based on a *global* criterion for solving a problem called AND-Quant, which generalizes the problem of reporting whether a given path is safe.

We present the first *local* characterization of safe paths for flow decompositions in directed acyclic graphs, giving a practical algorithm for finding the *complete* set of safe paths. We also evaluated our algorithm against the trivial safe algorithms (unitigs, extended unitigs) and a popular heuristic (greedy-width) for flow decomposition on RNA transcripts datasets. Despite maintaining perfect precision our algorithm reports $\approx 50\%$ higher coverage over trivial safe algorithms. Though greedy-width reports better coverage, it has significantly lower precision on complex graphs. On a unified metric (F-Score) of coverage and precision, our algorithm outperforms greedy-width by $\approx 20\%$, when the evaluated dataset has significant number of complex graphs. Also, it has superior time $(3-5\times)$ and space efficiency $(1.2-2.2\times)$, resulting in a better and more practical approach for bioinformatics applications of flow decomposition.

**Keywords:** safety · flow decomposition · DAGs · RNA assembly

* We thank Romeo Rizzi and Edin Husić for helpful discussions. This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO) and partially by the Academy of Finland (grants No. 322595, 328877) and the US NSF (award 1759522). The full version of the paper is available at [15].

## 1 Introduction

Network flows are a central topic in computer science, that define problems with countless practical applications. Assuming that the flow network has a unique source $s$ and a unique sink $t$, every flow can be decomposed into a collection of weighted $s$-$t$ paths and cycles [11]; for directed acyclic graphs (DAGs), such a decomposition contains only paths. Such a path (and cycle) view of a flow is used to optimally route information or goods from $s$ to $t$, where flow decomposition is a key step in problems such as network routing [13] and transportation [26]. Finding the decomposition with the minimum number of paths and *possibly* cycles (or *minimum flow decomposition*) is NP-hard, even for a DAG [37]. On the theoretical side, this hardness result led to research on approximation algorithms [13,30], and FPT algorithms [17,34]. On the practical side, many approaches employ a standard *greedy-width* heuristic [37], of repeatedly removing an $s$-$t$ path carrying the most flow. Another pseudo-polynomial-time heuristic called *Catfish* [32] tries to iteratively simplify the graph so that smaller decompositions can be found.

However, for a flow network built by superimposing a set of weighted paths, and one may seek the decomposition corresponding to that set of weighted paths. Such a decomposition is used by the more recent and prominent application of reconstructing biological sequences (*RNA transcripts* [35,34,40] or *viral quasi-species genomes* [5,4]). Each flow path represents a reconstructed sequence, and so a different set of flow paths encodes a different set of biological sequences, which may differ from the real ones. If there are multiple optimal solutions, then the reconstructed sequences may not match the original ones, and thus be incorrect. While many popular multiassembly tools use minimum flow decompositions, Williams et al. [41] reported that in an error-free transcript dataset 20% of human genes admit multiple minimum flow decomposition solutions.

### 1.1 Safety Framework for Addressing Multiple Solutions

Motivated by such an RNA assembly application, Ma et al. [20] were the first to address the issue of multiple solutions to the flow decomposition problem under a probabilistic framework. Later, they [21] solve a problem (*AND-Quant*), which, in particular, leads to a quadratic-time algorithm for the following problem: given a flow in a DAG, and edges $e_1, e_2, \ldots, e_k$, decide if in *every* flow decomposition there is always a decomposed flow path passing through all of $e_1, e_2, \ldots, e_k$. Thus, by taking the edges $e_1, e_2, \ldots, e_k$ to be a path $P$, the AND-Quant problem can decide if $P$ (i.e., a given biological sequence) appears in all flow decompositions. This indicates that $P$ is likely part of some original RNA transcript.

We build upon the AND-Quant problem, by addressing the flow decomposition problem under the *safety* framework [36], first introduced for genome assembly. For a problem admitting multiple solutions, a partial solution is said to be *safe* if it appears in all solutions to the problem. For example, a path $P$ is safe for the flow decomposition problem, if for *every* flow decomposition into paths $\mathcal{P}$, it holds that $P$ is a subpath of some path in $\mathcal{P}$. Further, $P$ is called *w-safe* if in *every* flow decomposition, $P$ is a subpath of some weighted path(s)

in $\mathcal{P}$ whose total weight is at least $w$. Bioinformatics applications [35,32,17] commonly use a minimum cardinality path decomposition (or path cover [19]). We consider *any* flow decomposition as a valid solution, not only the ones of minimum cardinality, which is motivated by both theory and practice. On the one hand, since minimum-cardinality flow decomposition is NP-hard [37], we believe that finding its safe paths is also intractable. On the other hand, given the issues with sequencing data, practical methods usually incorporate different variations of the minimality criterion [5,4]. Thus, safe paths for *all* flow decompositions are likely correct for many practical variations of the flow decomposition problem.

Safety has precursors in combinatorial optimization, as *persistency*. Costa [10] studied the persistent edges in all maximum bipartite matchings. Incidentally, for the maximum flow problem persistent edges always having a non-zero flow value in any maximum flow solution were studied [9]. In bioinformatics, safety has been previously studied for the genome assembly problem which at its core solves the problem of computing arc-covering walks on the assembly graph. Again since the problem admits multiple solutions where only one is correct, practical genome assemblers output only those solutions likely to be correct. The prominent approach dating back to 1995 [14] is to compute trivially correct *unitigs* (having internal nodes with *unit* indegree and unit outdegree), which can be computed in linear time. Unitigs were generalised first in [29], and later [23,16] to be *extended* by adding their unique incoming and outgoing paths. These *extended unitigs*, though safe, are not guaranteed to report *everything* that can be correctly assembled, presenting an important open question [25] about the *assembly limit* (if any). This was finally resolved by Tomescu and Medvedev [36] for a specific genome assembly formulation (single circular walk) by introducing *safe and complete* algorithms, which report everything that is theoretically reported as safe. Its running time was later optimized in [7] and [8]. Safe and complete algorithms were also studied by Acosta et al. [1] under a different genome assembly formulation of multiple circular walks. Recently, Cáceres et al. [6] studied safe and complete algorithms for path covers in an application on RNA Assembly.

### 1.2 Safety in Flow Decomposition for RNA Assembly

In bioinformatics, flow decomposition is prominently used in RNA transcript assembly, which is described as follows. In complex organisms, a gene may produce multiple RNA molecules (*RNA transcripts*, i.e., strings over an alphabet of four characters), each having a different abundance. Given a sample, one can partially read the RNA transcripts and find their abundances using *high-throughput sequencing* [38]. This technology produces short overlapping substrings of the RNA transcripts. The main approach for recovering the RNA transcripts from such data is to build an edge-weighted DAG from these fragments, then to transform the weights into flow values by various optimization criteria, and finally to decompose the resulting flow into an "optimal" set of weighted paths (i.e., the RNA transcripts and their abundances in the sample) [22]. A common strategy for choosing the optimal set of weighted paths is to look for the parsimonious solution, i.e., the solution with the fewest paths. Since this problem is NP-hard,

in practice many tools use the popular *greedy-width* heuristic [35,28]. Greedy-width is also used in the assemblers for the related problem of viral quasispecies assembly [4]. Further, some tools attempt to incorporate additional information into the flow decomposition process, such as by using longer reads or super reads [28,41]. Despite the large number of tools and methods that have been developed for RNA transcript assembly, there is no method that consistently reports the correct set of transcripts [28,42]. This suggests that the addressing the problem under the safety framework may be a promising approach. However, while a safe and complete solution clearly gives the maximally reportable correct solution, it is significant to evaluate whether such a solution covers a large part of the true transcript, to be useful in practice. A possible application of such partial and reliable solution is to consider them as constrains (see e.g. [41]) of real RNA transcript assemblers, to guide the assembly process of such heuristics. Another possible application could be to evaluate the accuracy of assemblers: does the output of the assembler include the safe and complete solution?.

### 1.3 Our Results

Our contributions can be succinctly described as follows.

1. **Simple local characterization and optimal verification algorithm**: We characterize a safe path $P$ using its local property called *excess flow*.

   **Theorem 1.** *For $w > 0$, a path $P$ is $w$-safe iff its excess flow $f_P \geq w$.*

   The previous work [21] on AND-Quant describes a *global* characterization using the maximum flow of the entire graph transformed according to $P$, requiring $O(mn)$ time. Instead, the excess flow is a *local* property of $P$ which is computable in time linear in the length of $P$. This also directly gives a simple verification algorithm which is optimal.

   **Theorem 2.** *Given a flow graph (DAG) having $n$ vertices and $m$ edges, it can be preprocessed in $O(m)$ time to verify the safety of a path $P$ in $O(|P|) = O(n)$ time.*

2. **Simple enumeration algorithm**: The above characterization also results in a simple algorithm for reporting all maximal safe paths by using an arbitrary flow decomposition of the graph.

   **Theorem 3.** *Given a flow graph (DAG) having $n$ vertices and $m$ edges, all its maximal safe paths can be reported in $O(|\mathcal{P}_f|) = O(mn)$ time, where $\mathcal{P}_f$ is some flow decomposition.*

   This approach starts with a candidate solution and uses the characterization on its subpaths in an efficient manner (a similar approach was previously used by [10,1,6]).

3. **Empirically improved approach for RNA assembly:** On simulated RNA splice graphs, safe and complete paths for flow decomposition provide precise RNA assemblies while covering most of RNA transcripts. They have $\approx 50\%$ better coverage over previous notions of safe paths, while maintaining the perfect precision ensured by safety. Further, for the combined metric of coverage and precision (F-Score), they outperform the popular greedy-width heuristic significantly ($\approx 20\%$) and previous safety algorithms appreciably ($\approx 13\%$). Though our approach takes $1.2 - 2.5\times$ time than the previous safety algorithms requiring equivalent memory, the greedy-width approach takes roughly $3 - 5\times$ time and $1.2 - 2.2\times$ memory than our approach. The significance of our approach in quality parameters increases with the increase in complex graph instances in the dataset, with significantly better performance over greedy-width, without losing a lot over previous safe algorithms.

## 2 Preliminaries and Notations

We consider a DAG $G = (V, E)$ with $n$ vertices and $m$ edges, where each edge $e$ has a positive flow (or *weight*) $f(e)$ passing through it. We assume the graph is connected and hence $m \geq n$. For each vertex $u$, $f_{in}(u)$ and $f_{out}(u)$ denote the total flow on its incoming edges and outgoing edges, respectively. A vertex $v$ is called a *source* if $f_{in}(v) = 0$ and a *sink* if $f_{out}(v) = 0$. Every other vertex $v$ satisfies the *conservation of flow* $f_{in}(v) = f_{out}(v)$, making the graph a *flow graph*. For a path $P$, $|P|$ denotes the number of its edges. For a set of paths $\mathcal{P} = \{P_1, \cdots, P_k\}$ we denote its total size (number of edges) by $|\mathcal{P}| = |P_1| + \cdots + |P_k|$.

For any flow graph (DAG), its *flow decomposition* is a set of weighted *paths* $\mathcal{P}_f$ such that the flow on each edge of the flow graph equals the sum of the weights of the paths containing the edge. A flow decomposition of a graph can be computed in $O(|\mathcal{P}_f|) = O(mn)$ time using the simple path decomposition algorithm [3]. A path $P$ is called *$w$-safe* if, in every possible flow decomposition, $P$ is a subpath of some paths in $\mathcal{P}_f$ whose total weight is at least $w$. If $P$ is $w$-safe with $w > 0$, we call $P$ a *safe flow path*, or simply *safe path*. Intuitively, for any edge $e$ with non-zero flow, we consider *where did the flow on $e$ come from?* We would like to report all the maximal paths ending with $e$ along which some $w > 0$ weight always "flows" to $e$ (see Figure 1). A safe path is *left maximal* (or *right maximal*) if extending it to the left (or right) with any edge makes it unsafe (i.e. not safe). A safe path is *maximal* if it is both left and right maximal. A set of safe paths is called *complete* if it consists of *all* the maximal safe paths.

Some previous notions of safety for other problems also naturally extend to the flow decomposition problem as follows. The paths having internal nodes with unit indegree and unit outdegree are called *unitigs* [14], which are trivially safe because every source-to-sink path which passes through an edge of unitig, also passes through the entire unitig contiguously. Further, a unitig can naturally be *extended* to include its unique incoming path (having nodes with unit indegree), and its unique outgoing path (having nodes with unit outdegree). This maximal extension of a unitig is called the *extended unitig* [23,16], which is similarly safe.

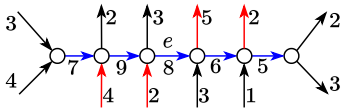



Fig. 1: The prefix of the path (blue) up to $e$ contributes at least 2 units of flow to $e$, as the rest may enter the path by the edges (red) with flow 4 and 2. Similarly, the suffix of the path (blue) from $e$ maintains at least 1 unit of flow from $e$, as the rest may exit the path from the edges (red) with flow 5 and 2. Both these safe paths are *maximal* as they cannot be extended left or right.

For some graphs the above notions already define the safety of flow decomposition *completely*. Millani et al. [24] defined a class of DAGs called *funnels*, where every source-to-sink path is uniquely identifiable by at least one edge, which is not used by any other source-to-sink path. Considering such an edge as a trivial unitig (having a single edge), its extended unitig is exactly the corresponding source-to-sink path, making it safe. Thus, in a funnel all the source-to-sink paths are naturally safe and hence trivially complete. Moreover, it implies that a funnel has a unique flow decomposition, making the problem trivial for funnel instances.

However, for non-funnel graphs unitigs and extended unitigs are safe but potentially not complete. Note that both unitigs and extended unitigs are also safe for problems dealing with unweighted graphs (eg. path cover). Hence, they do not use the flows on the edges of the graph, potentially missing some paths that are safe for flow decomposition but not for problems like path cover.

## 3 Characterization of Safe and Complete Paths

Safety of a path can be characterized by its *excess flow* defined as follows.

**Definition 1 (Excess flow).** Excess flow $f_P$ *of a path* $P = \{u_1, u_2, ..., u_k\}$ *is*

$$f_P = f(u_1, u_2) - \sum_{\substack{u_i \in \{u_2,\dots,u_{k-1}\} \\ v \neq u_{i+1}}} f(u_i, v) = f(u_{k-1}, u_k) - \sum_{\substack{u_i \in \{u_2,\dots,u_{k-1}\} \\ v \neq u_{i-1}}} f(v, u_i)$$

*the former and later formulations are* diverging *and* converging*, respectively.*

*Remark 1.* Alternatively, the converging and diverging formulations are

$$f_P = \sum_{i=1}^{k-1} f(u_i, u_{i+1}) - \sum_{i=2}^{k-1} f_{out}(u_i) = \sum_{i=1}^{k-1} f(u_i, u_{i+1}) - \sum_{i=2}^{k-1} f_{in}(u_i).$$

The converging and diverging formulations are equivalent by the conservation of flow on internal vertices. The idea behind excess flow $f_P$ (see Figure 2) is that even in the worst case, the maximum *leakage* , or the flow leaving (or entering) $P$

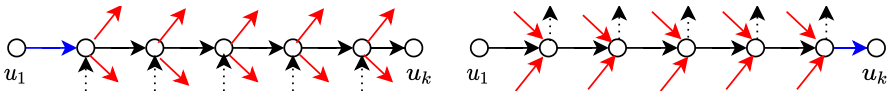



Fig. 2: The excess flow of a path $P$ (left) is the incoming flow (blue) that necessarily pass through the whole $P$ despite the flow (red) leaving $P$ at its internal vertices. Analogously (right), it is the outgoing flow (blue) that necessarily came through the whole $P$ despite the flow (red) entering $P$ at its internal vertices.

at the internal nodes, is the sum of the flow on the outgoing (or incoming) edges of the internal nodes of $P$, that are not in $P$. Hence, if the value of incoming flow (or outgoing flow) is higher than this maximum leakage, then this excess value $f_P$ necessarily passes through the entire $P$. The following results give the simple characterization and an additional property (see [15] for proof) of safe paths.

**Theorem 1.** *For $w > 0$, a path $P$ is $w$-safe iff its excess flow $f_P \geq w$.*

*Proof.* The excess flow $f_P$ of a path $P$ trivially makes it $w \leq f_P$-safe by definition. If $f_P < w$, we can prove that $P$ is not $w$-safe by modifying any flow decomposition having $w$ flow on $P$ to leave only $f_P$ flow (or 0, if $f_P < 0$) on $P$ as follows. In Figure 2 (diverging), consider a flow path $P'$ entering $P$ through edge $e_1$ (except first edge (blue)) and leaving $P$ at an edge $e_2$ (red) except last edge of $P$. Since $f_P < w$, it is not possible that every path leaving $P$ using a red edge starts at the first blue edge (by definition of $f_P$), hence $P'$ always exists. We modify $P'$ by using flow on $P$ to form two paths, which enter from $e_1$ and leave at the last edge, and which enter from the first edge and leave at $e_2$. We can repeat such modifications until flow on $P$ is $f_P$ (or 0, if $f_P < 0$) due to conservation of flow. Additionally, for a path to be safe, it must hold that $w > 0$.

**Lemma 1.** *Adding an edge $(u, v)$ to the start or the end of a path in the flow graph, reduces its excess flow by $f_{in}(v) - f(u, v)$, or $f_{out}(u) - f(u, v)$, respectively.*

## 4 Simple Verification and Enumeration Algorithms

The characterization of a safe path in a flow graph (Theorem 1) can be directly adapted to simple algorithms for verification and enumeration of all maximal safe paths. We preprocess the graph to compute the incoming flow $f_{in}(u)$ and outgoing flow $f_{out}(u)$ for each vertex $u$ in $O(m)$ time. Using Remark 1 we can verify if a path $P$ is safe in $O(|P|) = O(n)$ time, proving the following theorem.

**Theorem 2.** *Given a flow graph (DAG) having $n$ vertices and $m$ edges, it can be preprocessed in $O(m)$ time to verify the safety of a path $P$ in $O(|P|) = O(n)$ time.*

For reporting the maximal safe paths we use a candidate decomposition of the flow into paths, and verify the safety of its subpaths using the characterization and a scan with the two-pointer approach. The candidate flow decomposition can be computed in $O(mn)$ time using the classical flow decomposition algorithm [11] resulting in $O(m)$ paths $\mathcal{P}_f$ each of $O(n)$ length. Now, we use the two-pointer scan along each path $P \in \mathcal{P}_f$ as follows. We start with the subpath containing the first two edges of the path $P$. We compute its excess flow $f$, and if $f > 0$ we append the next edge to the path on the right and incrementally compute its excess flow by Lemma 1. Otherwise, if $f \leq 0$ we remove the first edge of the path from the left and incrementally compute the excess flow similarly by Lemma 1 (removing an edge $(u, v)$ would conversely modify the flow by $f_{in}(v) - f(u, v)$). We stop when the end of $P$ is reached with a positive excess flow.

The excess flow can be updated in $O(1)$ time when adding an edge to the subpath on the right or removing an edge from the left. If the excess flow of a subpath $P'$ is positive and on appending it with the next edge it ceases to be positive, we report $P'$ as a maximal safe path by reporting only its two indices on the path $P$. Thus, given a path of length $O(n)$, all its maximal safe paths can be reported in $O(n)$ time, and hence require total $O(mn)$ time for the $O(m)$ paths in the flow decomposition $\mathcal{P}_f$, resulting in the following theorem.

**Theorem 3.** *Given a flow graph (DAG) having $n$ vertices and $m$ edges, all its maximal safe paths can be reported in $O(|\mathcal{P}_f|) = O(mn)$ time, where $\mathcal{P}_f$ is some flow decomposition.*

## 5 Experimental Evaluation

We now evaluate the performance of our safe and complete algorithm by comparing it with the most promising algorithms for flow decomposition. Since the performance of various algorithms closely depend on the input graphs, we consider some practically relevant datasets to evaluate their true impact. As the most significant application of flow decomposition derives from RNA assembly, we consider the flow networks extracted as splice graphs of simulated RNA-Seq experiments. That is, starting from a set of RNA transcripts, we simulate their expression levels and superimpose the transcripts to create a flow graph. Evaluating our approach in such *perfect* scenario allows us to remove the biases introduced by real RNA-Seq experiments [33] and focus the features offered by the each technique instead. Further, the performance of algorithms also closely depend on the complexity $k$ of a graph, that we measure as the number of paths in the ground truth decomposition of the graph. Thus, we present our results with regards to the complexity $k$ of the input graph instances.

We first investigate the practical significance of *safety* by comparing our safe solution to a popularly used flow decomposition heuristic that is also scalable. The greedy-width [37] heuristic decomposes the flow by sequentially selecting the heaviest possible path, resulting in a simple algorithm that is both scalable and performs well in practice. However, any flow decomposition algorithm may not

always report the ground truth paths that originally built the instance of the flow graph. Thus, it is important to measure the reported solution using a *precision* metric which evaluates the correctness of the solution. We thus investigate how the precision of greedy-width varies particularly as the value of $k$ increases.

We then investigate the practical significance of *completeness* as reported by our solution, over the previously known safe solutions as reported by unitigs and extended unitigs (recall Section 2). Note that every safe solution would always score perfectly in a precision metric by definition. Hence, all safe solutions would always outperform greedy-width (or any flow decomposition algorithm) in precision metrics. However, this perfect precision comes at the cost of the amount of the solution that is reported. Intuitively, this can be measured using some *coverage* metrics which describe how much of the ground truth sequence is included in the reported paths. Note that any flow decomposition algorithm will perform better than any safe algorithm by definition, as the safe paths are always subpaths of the paths reported by any flow decomposition algorithm. Further, the extended unitigs would clearly outperform unitigs, and our safe paths would clearly outperform both unitigs and extended unitigs. We thus investigate how the coverage of various algorithms varies with respect to the greedy-width particularly as the value of $k$ increases.

Finally, to understand the overall impact of different algorithms, where safe algorithms as compared to greedy-width clearly outperform in precision metrics and underperform in coverage metrics, we address both coverage and precision measures using a single metric, i.e., F-score. We thus investigate the variation in F-score over different values of $k$. In addition, to understand the practical utility of the algorithms we also investigate their performance measures in terms of running time and space requirements.

### 5.1 Datasets

We consider two RNA transcripts datasets, generated based on approach of Shao et al. [32]. They create "perfect" flow graphs where the true set of transcripts and abundances is always a flow decomposition of the graph (hence satisfy conservation of flow). They start with this flow decomposition and create the input instances by superimposing them into a single graph, adding a single source $s$ (and sink $t$) with an edge to the beginning (and end) of each transcript.

*Funnel instances:* In funnels [24] all paths are safe and the problem is trivial (recall Section 2). Our evaluation thus ignores these trivial funnel instances. For the sake of completeness we address the funnels in our full paper [15].

*Catfish dataset:* We consider the dataset first used by Shao and Kingsford [32], which includes 100 simulated human transcriptomes for human, mouse, and zebrafish using Flux-Simulator [12]. Additionally, it includes 1,000 experiments from the Sequence Read Archive, with simulated abundances for transcripts using Salmon [27]. In both cases, the weighted transcripts are superimposed to build splice graphs as described above. This dataset has also been used in other

flow decomposition benchmarking studies [17,41]. There are 17,335,407 graphs in total in this dataset, of which 8,301,682 are non-trivial (47.89%). However, in this dataset the details about the number of bases on each node (exons or pseudo-exons) are omitted, which results in an incomplete measure of coverage and precision. Moreover, this dataset has negligible complex graph instances (having large $k$). Hence, we do not include its evaluation in the main paper, rather defer it to the full paper [15] for the sake of completeness.

*Reference-Sim dataset:* We consider a dataset [39] containing a single simulated transcriptome as follows. For each transcript on the positive strand in the GRCh.104 *homo sapiens* reference genome, it samples a value from the lognormal distribution with mean and variance both equal to $-4$, as done in the default settings of RNASeqReadSimulator [18]. It then multiplies the resulting values by 1000 and round to the nearest integer. Then it excludes any transcript that is rounded to 0. There are 17,941 total graphs in this dataset, of which 10,323 are non-trivial (57.54%). In this dataset, we also have access to the genomic coordinates (and hence number of bases) represented by nodes, allowing us to compute more practically relevant coverage and precision metrics.

### 5.2 Evaluation Metrics

All metrics are computed in terms of bases for the Reference-Sim dataset. However, since the Catfish dataset omits the base information its metrics are computed in terms of exons or pseudo-exons (vertices in the flow graph). For every algorithm, $R$ denotes a reported path (for Catfish) or a reported safe subpath (for unitigs, extended unitigs, and safe complete) of the solution. In addition, $T$ denotes a path in the set of ground truth transcripts provided in the dataset. For each graph, we compute the following metrics which were also used earlier by [6] for safety in constrained path covers:

*Weighted precision:* We classify a reported path $R$ as *correct* if $R$ is a subpath of some ground truth transcript $T$ of the flow graph. Weighted precision is the total length of correctly reported paths divided by the total length of reported paths. The commonly used precision metric [28,31] for measuring the accuracy of RNA assembly methods considers only those paths as correct which are (almost) exactly contained in the ground truth decomposition. Further, the precision is computed as the number of correctly reported paths divided by the total reported paths. However, since all the safe algorithms reports (possibly) partial transcripts, we use subpaths instead of (almost) exactly same paths. To highlight *how much* is reported correctly instead of *how many*, we use weighted precision to give a better score for longer correctly reported paths.

*Maximum relative coverage:* Given a ground truth transcript $T$ and a reported path $R$, we define a *segment* of $R$ inside $T$ as a maximal subpath of $R$ that is also subpath of $T$. We define the maximum relative coverage of a ground truth transcript as the length of the longest segment of a reported

path inside $T$, divided by the length of $T$. The corresponding value for the entire graph is the average of the values over all $T$. While it is common in the literature [28,31] to report *sensitivity* (the proportion of ground truth transcripts that are correctly predicted), we measure correctness based on coverage since all the safe algorithms report paths that (possibly) do not cover an entire transcript.

*F-score:* The standard measure to combine precision and sensitivity is using an *F-score*, which is the harmonic mean of the two. In our evaluation we correspondingly use the weighted precision and the maximum relative coverage for computing the F-score.

### 5.3 Implementation and Environment Details

We evaluate the following algorithms in our experiments.

*Unitigs:* It computes the unitigs, by considering each unvisited edge in the topological order and extending it towards the right as long as the internal nodes have unit indegree and unit outdegree. The result ignores single edges.

*ExtUnitigs:* It computes the extended unitigs, by considering each unitig and single edges, and extending it towards the left and right as long as the internal nodes have unit indegree and unit outdegree, respectively.

*Safe&Comp:* It computes the safe and complete paths for flow decomposition using our enumeration algorithm described in Section 4. Since the metrics evaluation scripts uses each safe path individually (similar to other algorithms), we output all safe paths completely which requires output size (and hence time) of $O(mn^2)$ instead of $O(mn)$ as stated in Theorem 1.

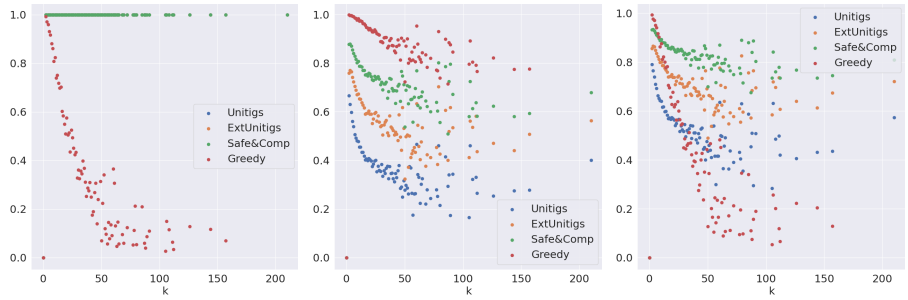
*Greedy:* It computes the greedy-width heuristic using Catfish [32] with the `-a greedy` parameter.

All algorithms are implemented in C++, whereas the scripts for evaluating metrics are implemented in Python. The Unitigs, ExtUnitigs, and Safe&Comp implementations use optimization level 3 of GNU C++ (compiled with $-O3$ flag), whereas the Greedy uses the optimizations of the Catfish pipeline. The Unitigs, ExtUnitigs, and Safe&Comp additionally require a post processing step using Aho Corasick Trie [2] for removing duplicates, and prefix/suffixes to make the set of safe paths minimal. However, the time and memory requirements are evaluated considering only the algorithm, and not post processing and metric evaluations which are not optimized. All performances were evaluated on a laptop using a single core (i5-8265U CPU 1.60GHZ) having 15.3GB memory. The source code of our project is available on Github [4] under GNU Genral Public License v3 license.

### 5.4 Results

We first evaluate the significance of *safety* among the reported solution. Figure 3a compares the weighted precision of all the algorithms on the ReferenceSim dataset distributed over $k$. All the safe algorithms clearly report perfect

[4] https://github.com/algbio/flow-decomposition-safety




(a) Weighted Precision (b) Max Relative Coverage (c) F-Score

Fig. 3: Evaluation metrics on graphs w.r.t. $k$ for the Reference-Sim dataset.

precision as expected. However, the precision of the Greedy algorithm sharply declines with the increase in $k$, almost linearly to 30% for $k = 35$. This may be explained by the sharp increase in the number of possible paths in graphs with increase in $k$, which can be used by any flow decomposition algorithm. Hence, the significance of safety becomes very prominent as $k$ increases .

We then evaluate the significance of *completeness* of the safe algorithms. Figure 3b compares the maximum relative coverage of all the algorithms on the Reference-Sim dataset distributed over $k$. As expected, Greedy outperforms all the other, followed by Safe&Comp, ExtUnitigs and Unitigs. However, note that as $k$ reaches 20 Safe&Comp, ExtUnitigs and Unitigs sharply fall to 75%, 60% and 40%, while Greedy maintains around 95% coverage. Overall, Safe&Comp is almost always $\approx 85 - 90\%$ of that of Greedy, whereas ExtUnitigs and Unitigs falls to 60% and 40% respectively. Hence, the Safe&Comp manages to maintain perfect precision without losing a lot on coverage, demonstrating the importance of *completeness* among the safe algorithms.

Figure 3c supports the above inference by evaluating the combined metric F-Score, where Safe&Comp dominates Unitigs and ExtUnitigs by definition. Safe&Comp also dominates Greedy as $k$ approaches 10. It is also important to note that both ExtUnitigs and Unitigs eventually dominate Greedy for a slightly larger value of $k > 20$ and $k > 30$, respectively. This shows the significance of considering Safe algorithms for complex graphs. However, the significance of the Safe&Comp as the number of graphs with such higher complexities also reduces sharply (see full paper [15]).

Hence, we evaluate a summary of the above results averaged over all graphs irrespective of $k$. Table 1 summarizes the evaluation metrics for all the algorithms for simple graphs ($k < 10$) and complex graphs ($k > 10$), and both. While on the simpler graphs Greedy dominates Safe&Comp mildly ($\approx 3\%$), for complex graphs it is dominated significantly ($\approx 20\%$) by Safe&Comp and appreciably ($\approx 8\%$) by ExtUnitigs. However, despite the larger ratio of simpler graphs, the collective F-score over all graphs is still ($\approx 4\%$) better for Safe&Comp over Greedy which signifies the applicability of Safe&Comp over Greedy.

| Graphs | Algorithm | Max. Coverage | Wt. Precision | F-Score |
|---|---|---|---|---|
| $k \geq 2$ (100%) | Unitigs | 0.51 | 1.00 | 0.66 |
| | ExtUnitigs | 0.69 | 1.00 | 0.81 |
| | Safe&Comp | 0.82 | 1.00 | 0.90 |
| | Greedy | 0.98 | 0.81 | 0.86 |
| $2 \leq k \leq 10$ (68%) | Unitigs | 0.55 | 1.00 | 0.70 |
| | ExtUnitigs | 0.73 | 1.00 | 0.84 |
| | Safe&Comp | 0.84 | 1.00 | 0.91 |
| | Greedy | 0.99 | 0.91 | 0.94 |
| $k > 10$ (32%) | Unitigs | 0.41 | 1.00 | 0.58 |
| | ExtUnitigs | 0.61 | 1.00 | 0.75 |
| | Safe&Comp | 0.76 | 1.00 | 0.86 |
| | Greedy | 0.95 | 0.60 | 0.69 |

Table 1: Summary of evaluation metrics for the Reference-Sim dataset.

| Algorithm | Reference-Sim | | Catfish | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Human 25.6MB | | Zebrafish 122MB | | Mouse 137MB | | Human 157MB | | Human (salmon) 2.5GB | |
| | Time | Mem | Time | Mem | Time | Mem | Time | Mem | Time | Mem |
| Unitigs | 0.68 | 3.58 | 13.82 | 3.51 | 15.62 | 3.53 | 18.22 | 3.54 | 303.72 | 3.66 |
| ExtUnitigs | 0.99 | 3.63 | 18.31 | 3.52 | 20.87 | 3.57 | 23.64 | 3.56 | 404.50 | 3.68 |
| Safe&Comp | 2.56 | 4.47 | 20.17 | 3.56 | 25.76 | 3.66 | 28.59 | 3.54 | 667.27 | 3.84 |
| Greedy | 7.71 | 4.88 | 108.30 | 6.00 | 127.38 | 6.29 | 148.46 | 6.34 | 2684.30 | 8.47 |

Table 2: Time (s) and Memory (MB) taken by different algorithms on datasets.

Finally, we evaluate the applicability of the above algorithms in practice, by comparing their running time and peak memory requirements. Since all the algorithms are implemented in the same language (C++) and evaluated on the same machine, it is reasonable to directly compare these measures. In Table 2, we see that Unitigs clearly are the fastest, where ExtUnitigs takes roughly $1.3 - 1.5\times$ time. Safe&Comp takes upto roughly $1.2 - 2.5\times$ time than ExtUnitigs, and Greedy requires roughly $3 - 5\times$ time than Safe&Comp. The peak memory requirements of the safe algorithms are very close (within 5%-25%), whereas Greedy requires roughly $1.1 - 2.2\times$ more memory than Safe&Comp. Overall, for the performance measures Safe&Comp shows a significant improvement over Greedy, without losing a lot over the trivial algorithms.

## 6 Conclusion

We study the flow decomposition in DAGs under the Safe and Complete paradigm, which has numerous applications including the more prominent multi-assembly of biological sequences. Previous work characterized such paths (and their generalizations) using a global criterion. Instead, we present a simpler characterization based on a more efficiently computable local criterion, which is directly adapted

into an optimal verification algorithm, and a simple enumeration algorithm. Intuitively, it is a *weighted* adaptation of *extended unitigs* which is a prominent approach for computing safe paths.

Our experiments show that our algorithm outperform the popularly used greedy-width heuristic for RNA assembly instances having significant complex graph instances, both on quality (F-score) and performance (running time and memory) parameters. On simple graphs, Greedy outperforms Safe&Comp and Safe&Comp outperforms ExtUnitigs mildly ($\approx 3-5\%$). However, on complex graphs, Safe&Comp outperforms Greedy significantly ($\approx 20\%$) and ExtUnitigs appreciably ($\approx 13\%$). While the Reference-Sim dataset shows the overall dominance of Safe&Comp since complex graphs are appreciable (32%), Greedy dominates Safe&Comp in Catfish dataset since complex graphs are negligible ($\approx 2\%$). Another significant reason for the dominance of Greedy over Safe&Comp on Catfish datasets is the absence of base information on nodes (see full paper [15]). Hence, the importance of Safe&Comp algorithms increases with the increase in complex graph instances in the dataset, and prominently when we consider information about the genetic information represented by each node. In terms of performance, ExtUnitigs are $1.3-1.5\times$ slower than the fastest approach (Unitigs), while Safe&Comp further takes roughly $1.2-2.5\times$ time than ExtUnitigs, both requiring equivalent memory. However, Greedy requires roughly $3-5\times$ time and $1.1-2.2\times$ memory than Safe&Comp. Overall, Safe&Comp performs significantly better than Greedy, without losing a lot over the trivial algorithms.

## References

1. Acosta, N.O., Mäkinen, V., Tomescu, A.I.: A safe and complete algorithm for metagenomic assembly. Algorithms for Molecular Biology **13**(1), 3:1–3:12 (2018). https://doi.org/10.1186/s13015-018-0122-7
2. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Commun. ACM **18**(6), 333–340 (1975). https://doi.org/10.1145/360825.360855
3. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network flows - theory, algorithms and applications. Prentice Hall (1993)
4. Baaijens, J.A., der Roest, B.V., Köster, J., Stougie, L., Schönhuth, A.: Full-length de novo viral quasispecies assembly through variation graph construction. Bioinform. **35**(24), 5086–5094 (2019). https://doi.org/10.1093/bioinformatics/btz443
5. Baaijens, J.A., Stougie, L., Schönhuth, A.: Strain-aware assembly of genomes from mixed samples using flow variation graphs. In: Schwartz, R. (ed.) Research in Computational Molecular Biology - 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10-13, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12074, pp. 221–222. Springer (2020). https://doi.org/10.1007/978-3-030-45257-5_14
6. Caceres, M., Mumey, B., Husic, E., Rizzi, R., Cairo, M., Sahlin, K., Tomescu, A.I.I.: Safety in multi-assembly via paths appearing in all path covers of a DAG. IEEE/ACM Transactions on Computational Biology and Bioinformatics (2021)
7. Cairo, M., Medvedev, P., Acosta, N.O., Rizzi, R., Tomescu, A.I.: An Optimal $O(nm)$ Algorithm for Enumerating All Walks Common to All Closed Edge-

covering Walks of a Graph. ACM Trans. Algorithms **15**(4), 48:1–48:17 (2019). https://doi.org/10.1145/3341731
8. Cairo, M., Rizzi, R., Tomescu, A.I., Zirondelli, E.C.: Genome assembly, from practice to theory: Safe, complete and linear-time. In: Bansal, N., Merelli, E., Worrell, J. (eds.) 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference). LIPIcs, vol. 198, pp. 43:1–43:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
9. Cechlárová, K., Lacko, V.: Persistency in combinatorial optimization problems on matroids. Discret. Appl. Math. **110**(2-3), 121–132 (2001). https://doi.org/10.1016/S0166-218X(00)00279-1
10. Costa, M.C.: Persistency in maximum cardinality bipartite matchings. Operations Research Letters **15**(3), 143 – 149 (1994). https://doi.org/10.1016/0167-6377(94)90049-3
11. Ford, D.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, USA (2010)
12. Griebel, T., Zacher, B., Ribeca, P., Raineri, E., Lacroix, V., Guigó, R., Sammeth, M.: Modelling and simulating generic rna-seq experiments with the flux simulator. Nucleic acids research **40**(20), 10073–10083 (2012)
13. Hartman, T., Hassidim, A., Kaplan, H., Raz, D., Segalov, M.: How to split a flow? In: 2012 Proceedings IEEE INFOCOM. pp. 828–836. IEEE (2012)
14. Kececioglu, J.D., Myers, E.W.: Combinatorial algorithms for DNA sequence assembly. Algorithmica **13**(1/2), 7–51 (1995)
15. Khan, S., Kortelainen, M., Cáceres, M., Williams, L., Tomescu, A.I.: Safety and Completeness in Flow Decompositions for RNA Assembly. CoRR **abs/2201.10372** (2022)
16. Kingsford, C., Schatz, M.C., Pop, M.: Assembly complexity of prokaryotic genomes using short reads. BMC Bioinformatics **11**(1), 21 (2010)
17. Kloster, K., Kuinke, P., O'Brien, M.P., Reidl, F., Villaamil, F.S., Sullivan, B.D., van der Poel, A.: A practical fpt algorithm for flow decomposition and transcript assembly. In: 2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX). pp. 75–86. SIAM (2018)
18. Li, W.: RNASeqReadSimulator: a simple RNA-seq read simulator (2014)
19. Liu, R., Dickerson, J.: Strawberry: Fast and accurate genome-guided transcript reconstruction and quantification from rna-seq. PLoS computational biology **13**(11), e1005851 (2017)
20. Ma, C., Zheng, H., Kingsford, C.: Exact transcript quantification over splice graphs. In: Kingsford, C., Pisanti, N. (eds.) 20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference). LIPIcs, vol. 172, pp. 12:1–12:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.WABI.2020.12
21. Ma, C., Zheng, H., Kingsford, C.: Finding ranges of optimal transcript expression quantification in cases of non-identifiability. bioRxiv (2020). https://doi.org/10.1101/2019.12.13.875625, to appear at RECOMB 2021
22. Mäkinen, V., Belazzougui, D., Cunial, F., Tomescu, A.I.: Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing. Cambridge University Press (2015). https://doi.org/10.1017/CBO9781139940023
23. Medvedev, P., Georgiou, K., Myers, G., Brudno, M.: Computability of models for sequence assembly. In: WABI. pp. 289–301 (2007)
24. Millani, M.G., Molter, H., Niedermeier, R., Sorge, M.: Efficient algorithms for measuring the funnel-likeness of dags. Journal of Combinatorial Optimization **39**(1), 216–245 (2020)

25. Nagarajan, N., Pop, M.: Parametric complexity of sequence assembly: theory and applications to next generation sequencing. Journal of computational biology **16**(7), 897–908 (2009)
26. Olsen, N., Kliewer, N., Wolbeck, L.: 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 (2020). https://doi.org/10.1007/s10100-020-00705-6
27. Patro, R., Duggal, G., Kingsford, C.: Salmon: accurate, versatile and ultrafast quantification from rna-seq data using lightweight-alignment. BioRxiv p. 021592 (2015)
28. Pertea, M., Pertea, G.M., Antonescu, C.M., Chang, T.C., Mendell, J.T., Salzberg, S.L.: Stringtie enables improved reconstruction of a transcriptome from rna-seq reads. Nature biotechnology **33**(3), 290–295 (2015)
29. Pevzner, P.A., Tang, H., Waterman, M.S.: An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences **98**(17), 9748–9753 (2001)
30. Pieńkosz, K., Kołtyś, K.: Integral flow decomposition with minimum longest path length. European Journal of Operational Research **247**(2), 414–420 (2015)
31. Shao, M., Kingsford, C.: Accurate assembly of transcripts through phase-preserving graph decomposition. Nature biotechnology **35**(12), 1167–1169 (2017)
32. Shao, M., Kingsford, C.: Theory and a heuristic for the minimum path flow decomposition problem. IEEE/ACM Transactions on Computational Biology and Bioinformatics **16**(2), 658–670 (2017)
33. Srivastava, A., Malik, L., Sarkar, H., Zakeri, M., Almodaresi, F., Soneson, C., Love, M.I., Kingsford, C., Patro, R.: Alignment and mapping methodology influence transcript abundance estimation. Genome Biology **21**(1), 1–29 (2020)
34. Tomescu, A.I., Gagie, T., Popa, A., Rizzi, R., Kuosmanen, A., Mäkinen, V.: Explaining a weighted DAG with few paths for solving genome-guided multi-assembly. IEEE ACM Trans. Comput. Biol. Bioinform. **12**(6), 1345–1354 (2015). https://doi.org/10.1109/TCBB.2015.2418753
35. Tomescu, A.I., Kuosmanen, A., Rizzi, R., Mäkinen, V.: A novel min-cost flow method for estimating transcript expression with rna-seq. BMC bioinformatics **14**(S5), S15 (2013)
36. Tomescu, A.I., Medvedev, P.: Safe and complete contig assembly through omnitigs. Journal of Computational Biology **24**(6), 590–602 (2017), preliminary version appeared in RECOMB 2016.
37. Vatinlen, B., Chauvet, F., Chrétienne, P., Mahey, P.: Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. European Journal of Operational Research **185**(3), 1390–1401 (2008)
38. Wang, Z., Gerstein, M., Snyder, M.: RNA-Seq: a revolutionary tool for transcriptomics. Nature Reviews Genetics **10**(1), 57–63 (2009)
39. Williams, L.: Reference-sim (Nov 2021). https://doi.org/10.5281/zenodo.5646910
40. Williams, L., Reynolds, G., Mumey, B.: Rna transcript assembly using inexact flows. In: 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 1907–1914. IEEE (2019)
41. Williams, L., Tomescu, A., Mumey, B.M., et al.: Flow decomposition with subpath constraints. In: 21st International Workshop on Algorithms in Bioinformatics (WABI 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)
42. Yu, T., Mu, Z., Fang, Z., Liu, X., Gao, X., Liu, J.: Transborrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers. Genome research **30**(8), 1181–1190 (2020)