Accelerating ILP Solvers for Minimum Flow Decompositions Through Search Space and Dimensionality Reductions

Andreas Grigorjew
University of Helsinki, Finland

Fernando H. C. Dias
Aalto University, Espoo, Finland

Andrea Cracco
University of Verona, Italy

Romeo Rizzi
University of Verona, Italy

Alexandru I. Tomescu
University of Helsinki, Finland

Abstract

Given a flow network, the Minimum Flow Decomposition (MFD) problem is finding the smallest possible set of weighted paths whose superposition equals the flow. It is a classical, strongly NP-hard problem that is proven to be useful in RNA transcript assembly and applications outside of Bioinformatics.

We improve an existing ILP (Integer Linear Programming) model by Dias et al. [RECOMB 2022] for DAGs by decreasing the solver's search space using *solution safety* and several other optimizations. This results in a significant speedup compared to the original ILP, of up to 34× on average on the hardest instances. Moreover, we show that our optimizations apply also to MFD problem variants, resulting in speedups that go up to 219× on the hardest instances.

We also developed an ILP model of reduced dimensionality for an MFD variant in which the solution path weights are restricted to a given set. This model can find an optimal MFD solution for most instances, and overall, its accuracy significantly outperforms that of previous greedy algorithms while being up to an order of magnitude faster than our optimized ILP.

2012 ACM Subject Classification Theory of computation → Network flows; Applied computing → Bioinformatics

Keywords and phrases Flow decomposition, Integer Linear Programming, Safety, RNA-seq, RNA transcript assembly, isoform

Digital Object Identifier 10.4230/LIPIcs.SEA.2024.14

Supplementary Material *Software (Source Code, Datasets)*: `https://github.com/algbio/optimized-fd`, archived at `swh:1:dir:1e7c3ad9381a18899122a662b79914eb33cece80`

Funding 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 by the Research Council of Finland (grants No. 322595, 352821, 346968, 358744).

Acknowledgements We are very grateful to Manuel Cáceres for a very helpful discussion on sets of independent safe paths.

© Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu; licensed under Creative Commons License CC-BY 4.0
22nd International Symposium on Experimental Algorithms (SEA 2024).
Editor: Leo Liberti; Article No. 14; pp. 14:1–14:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Motivation. Minimum Flow Decomposition (MFD) is the problem of finding a minimum number of weighted paths to decompose a flow on a directed graph, such that the sum of weights of the paths crossing an edge is equal to the flow on that edge. We refer to this number as the size of the MFD. Hartman et al. [9] proved that MFD is NP-hard, even on directed acyclic graphs (DAGs), and even if the flow values are only in $\{1, 2, 4\}$.

MFD is a crucial tool in many applications, for example, in networking, where the paths represent traffic going through a sequence of routers. Minimizing the number of these paths reduces the amount of maintenance work, among other additional benefits [9]. In RNA sequence reconstruction problems, MFD is used by e.g. [16, 22, 6, 2, 21, 25] to decompose a *splice graph*, in which every node is a gene *exon*, and every weighted path is an RNA sequence that needs to be reconstructed. This splice graph can, for example, be constructed by aligning RNA-seq reads to a reference genome. MFD can also used to reconstruct viral quasispecies [1]. The input graphs in these applications are DAGs, and all previous work on MFD that we cite below is also on DAGs. As such, in this paper, all graphs are DAGs.

Moreover, MFD variants are of practical interest. For example, one practical strategy is to incorporate longer reads as *subpaths* potentially spanning more than two nodes [16, 19, 26, 8], which must appear in some solution path (i.e. reconstructed transcript or viral genome). Alternatively, the edge weights may not form an exact flow, due to sequencing errors and read mapping artifacts. An MFD variant adapted to this case is to consider intervals of edge weights instead, called *inexact flows* in [18, 25].

Related work. Given the complexity of the problem, it is common to use heuristic methods [20], for example, by using a greedy approach, iteratively removing the path with the largest currently available flow until the flow is fully decomposed. Cáceres et al. [17] showed that for DAGs this approach performs well, with an approximation ratio of $O(\log \textsc{val}(f))$ (where $\textsc{val}(f)$ is the total flow of the graph), only if the *width* of the graph (the minimum number of paths to cover all edges) does not increase in the process; otherwise it gives an exponentially worse result than the optimal with an approximation ratio of $\Omega(m/\log m)$.

Mumey et al. [14] gave an approximation algorithm on DAGs with ratio $\lambda^{\log\|f\|} \log\|f\|$, where λ is the longest source-to-sink path length and $\|f\|$ is the maximum-norm on the flow f (i.e., the largest flow value of the graph). Polynomial-time approximation algorithms with sublinear ratio in the size of the graph are not known, neither whether the problem is in APX, i.e. admits a constant-factor approximation algorithm.

Kloster et al. [11] proposed the first algorithm solving MFD optimally, which is linear-time fixed-parameter tractable (FPT), parameterized on the MFD size k. While it performs well for small k values, it quickly becomes unfeasible for larger k values, since the parameterized runtime grows exponentially on k with a degree 2 polynomial in the exponent. This is a limiting factor to the usability of such algorithms in practical applications.

In a recent work, Dias et al. [4] formulated an Integer Linear Problem (ILP) model for MFD on DAGs and showed that it performs faster on graphs with larger solution sizes than the FPT algorithm of Kloster et al. [11]. Due to this better performance of an ILP, it is natural to work on further optimizations. An additional strength of ILP formulations is their extensibility, since ILPs can easily be modified to handle other aspects of the applications, such as subpath constraints and inexact flows cited above, see [4].

Contributions. In this paper, we significantly optimize the ILP for MFD using the notion of *safe paths* for all flow decompositions in DAGs, first studied by Khan et al. [10]. More specifically, a path P is said to be *safe* if all flow decompositions (of any size) have at least one path containing P as a subpath. Clearly, safe paths must appear in any MFD. We show how they can be used to reduce the ILP search space by using them to fix many binary variables of the ILP. At a high level, this approach is along the same lines as for other NP-hard problems, for example, Bumpus et al. [3] analyzed *c-essential vertices*, as those vertices belonging to any c-approximate solution, for some specific graph problems whose solutions are sets of vertices. Clearly, c-essential vertices also belong to all optimal solutions, and can be simply removed from the graph when running e.g. an FPT algorithm (see [3]).

In our case, safe solutions are sets of *paths* (not single vertices), and it is non-trivial how to use them to simplify the input graph, or how to integrate them into a combinatorial algorithm (e.g., we cannot simply remove them, they could overlap, etc.). However, incorporating safe paths into an ILP is easily supported since they can be modelled by additional constraints, or by fixing some ILP variables. Moreover, in order to use more than one safe path, we observe that pairwise unreachable paths (they are not subpaths of a superpath) must be present in *different* MFD solution paths. As a fast heuristic to select a set of such paths of large total length, we use a reduction to a maximum weight antichain problem [12]. In Appendix B we show that a *maximum-length* set of pairwise unreachable safe paths can also be found in polynomial time, but this procedure is overall computationally more involved, and thus may be too expensive as a pre-processing step.

Furthermore, as observed by Cáceres et al. [17], the size of any edge antichain (like ours) is a lower bound on the MFD size. As such, we can also use this lower bound to check if a heuristic MFD solution size attains this lower bound, in which case it is optimal. To the best of our knowledge, this is the first time such a (non-trivial) lower bound is used in an MFD solver. Lastly, before running the ILP solver, we also simplify the input graph using the Y-to-V graph compaction, also used in the implementation of the FPT algorithm of Kloster et al. [11], but not used in the one of the ILP by Dias et al. [4]. Using all these optimizations, we experimentally show that we obtain significant speedups over the original ILP, of up to 34× on the harder inputs.

In addition, we also adapt these optimizations to the MFD variants considering inexact flows and subpath constraints, proving that such adaptations can easily be done also for MFD variants of practical interest. Furthermore, we also show (in Section 4) how the Y-to-V graph compaction can be modified for these two variants, an issue which has not been considered before in the literature. With these optimizations, on the hardest graphs among the instances with inexact flows, we obtain an impressive speedup of 219× compared to the original ILP for MFD with subpath constraints.

As a last contribution, we tackle an MFD variant where the weights of the solution paths must belong to a given set, initially defined by Kloster et al. [11]. We show that this problem on DAGs admits a simpler and thus faster ILP than the one for MFD. When the set of given weights consists of powers of 2 (up to the maximum flow value of any edge), this ILP provides a $\log ||f||$-factor approximation ratio for the original MFD problem. For practical applications, we show that if we also add the flow values of all edges to the given set of weights, then this ILP returns an MFD solution with a number of paths that is significantly closer to the minimum one than the state-of-the-art heuristic MFD solver Catfish [20], on instances with large MFD size. These are of particular interest, since our optimized solver runs in under 2 seconds on instances of solution size at most 10. The ILP for this problem variant is up to a further order of magnitude faster than our optimized ILP, and running in under 1 second on average.

2 Preliminaries

Definitions. We let $G = (V, E)$ be a graph and assume throughout the paper that G is a DAG (i.e. directed and acyclic). For simplicity we also assume that G has no parallel edges and that it has a single source node s and a single sink node t. We denote by $\mathbb{N}$ the natural numbers including 0 and by $\mathbb{Z}^+ = \mathbb{N} \setminus \{0\}$ the positive integers. A *u-v path* P is a sequence of edges going from the node u to v and its length $|P|$ is defined as its number of edges. Additionally, we identify paths P with functions $P : E \to \{0, 1\}$ where $P(e) = 1$ iff $e \in P$. We call two paths P_1, P_2 (and resp. edges) *independent*, if there exists no path P^* such that P_1 and P_2 are subpaths of P^*. The motivation for this definition is the fact that reachability between paths is closed under transitivity. If the DAG is a poset (i.e., E is closed under transitivity), then two edges (u, v) and (x, y) are independent if and only if the nodes v and x are independent or u and y are independent. For a path $P = (e_1, \ldots, e_{|P|})$ we denote by $P[\ell..r] := (e_\ell, \ldots, e_r)$ a subpath of length $r - \ell + 1$. A *flow network* is the tuple $G = (V, E, f)$, where $f : E \to \mathbb{Z}^+$ is a flow, i.e. a function that preserves the flow conservation $\sum_{(u,v)\in E} f_{uv} = \sum_{(v,w)\in E} f_{vw}$, $\forall v \in V \setminus \{s, t\}$.

By a *k-flow decomposition* of a flow f (see Figure 1) we denote a family of s-t paths $\mathcal{P} = (P_1, \ldots, P_k)$ with their associated weights $w = (w_1, \ldots, w_k) \in (\mathbb{Z}^+)^k$, such that:

$$\sum_{i\in\{1,\ldots,k\}} P_i(u, v)w_i = f_{uv}, \quad \forall (u, v) \in E. \tag{1}$$

In this paper, we focus on positive integer flow values, which is motivated by its application in RNA sequence reconstruction. Vatinlen et al. [24] proved that there is always a $(|E|-|V|+2)$-flow decomposition. The *Minimum Flow Decomposition (MFD)* problem asks for a k-flow decomposition with minimum k.

In case of imperfect data, Williams et al. [25] considered the problem variant *Minimum Inexact Flow Decomposition (MIFD)* where for each edge (u, v) we have a lower bound L_{uv} and an upper bound R_{uv}, and constraint (1) is changed to require that $\sum_{i\in\{1,\ldots,k\}} P_i(u, v)w_i \in [L_{uv}, R_{uv}]$, $\forall (u, v) \in E$. Another MFD variant of practical interest is *Minimum Flow Decomposition with Subpath Constraints (MFDSC)* [26]. The only difference with respect to MFD is that in the input we also have a set of paths $\mathcal{R}$, called *subpath constraints*, and it is required that every path in $\mathcal{R}$ is a subpath of at least one path of the decomposition.

(a) A flow network.

(b) A flow decomposition into paths of weights $(1, 1, 2, 3, 3)$.

Figure 1 Example of a flow network in (1a) and a decomposition of it into five s-t paths in (1b).

ILP formulations. We now review the ILP formulation for MFD given by Dias et al. [4], which decomposes the flow into k paths. Since the value for k is not known beforehand, one must iterate and increase this value until the ILP becomes feasible. To model each path P_i, $i \in \{1, \ldots, k\}$, binary variables x_{uvi} are introduced to represent each edge $(u, v) \in E$. We set

$x_{uvi} = P_i(u, v)$, i.e. $x_{uvi} = 1$ if $(u, v) \in P_i$ and otherwise $x_{uvi} = 0$. At the same time, each path is required to start from the source and end in the sink. For all the intermediary nodes of a path, a unit in-degree and a unit out-degree is required. Those requirements can be modeled by the following constraints. For all $v \in V$ and for all $i \in \{1, \ldots, k\}$:

$$\sum_{(u,v)\in E} x_{uvi} - \sum_{(v,u)\in E} x_{vui} = \begin{cases} 0, & \text{if } v \in V \setminus \{s, t\}, \\ 1, & \text{if } v = t, \\ -1, & \text{if } v = s, \end{cases} \tag{2}$$

Constraint (1) is then modeled as:

$$f_{uv} - \sum_{i\in\{1,\ldots,k\}} x_{uvi} w_i = 0, \quad \forall (u, v) \in E. \tag{3}$$

The above constraint contains the non-linear term $x_{uvi} w_i$ that standard linear solvers cannot solve. However, using basic linearization techniques, each such constraint can be replaced by three linear constraints, see [4] for details. For completeness, we give the full formulation in Appendix A.1.

Dias et al. [4] used the flexibility of this ILP formulation to easily model also the two variants MIFD and MFDSC. The former problem can easily be modeled in ILP by changing the (linearized version of) constraint (3) to state that $\sum_{i\in\{1,\ldots,k\}} x_{uvi} w_i$ belongs to the interval $[L_{uv}, R_{uv}]$. The latter problem can be modeled by introducing additional indicator variables $r_{ij} \in \{0, 1\}$ for every $i \in \{1, \ldots, k\}$, and every path $R_j \in \mathcal{R}$, modeling if the subpath j is contained in the i-th path of the flow decomposition, via the additional constraints:

$$\sum_{(u,v)\in R_j} x_{uvi} \geq |R_j| r_{ij}, \quad \forall i \in \{1, \ldots, k\}, \forall R_j \in \mathcal{R}, \tag{4}$$

$$\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq 1, \quad \forall R_j \in \mathcal{R}. \tag{5}$$

3 Reducing the ILP search space via safe paths

In this section, we optimize the ILP model for MFD presented above based on the notion of *solution safety* (see e.g. Tomescu and Medvedev [23]). Then, in Remarks 2 and 3, we discuss how this method can also be applied to the MIFD and MFDSC problems.

We call a path *safe* if it is part of every (not necessarily optimal) flow decomposition; that is, if it is a subpath of some path in every decomposition (not necessarily of minimum size). Khan et al. [10] characterized all safe paths via the notion of *excess flow*, which can be thought of as the flow value of the first edge of the path minus the flow values of the non-path edges out-going from the internal nodes of the path (i.e., the flow "leaking" from the path). Formally:

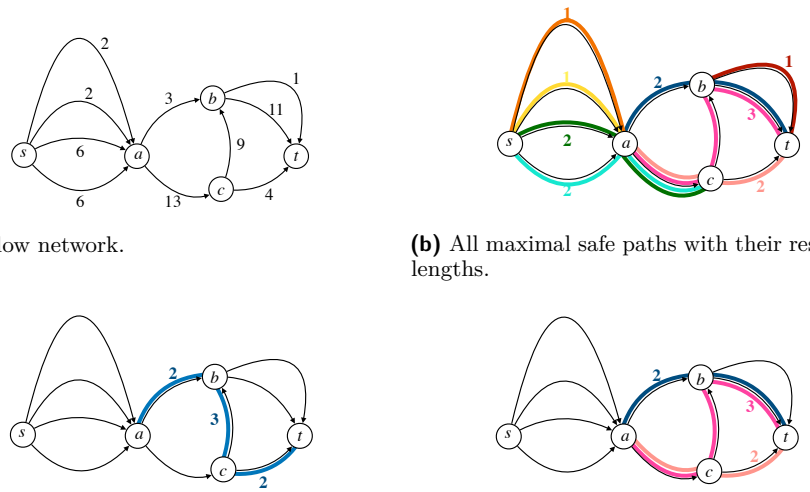
▶ **Lemma 1** ([10]). *For a path $P = (e_1, \ldots, e_{|P|})$ with $e_i = (u_i, u_{i+1})$, define its* excess flow*:*

$$f_P := f(e_1) - \sum_{\substack{(u_i,v)\in E \\ i\in\{2,\ldots,|P|\}, v\neq u_{i+1}}} f(u_i, v).$$

A path P is safe if and only if $f_P > 0$.

Since we assume the flow to be positive, every edge must be decomposed, and is thus (as a path of length 1) safe. Khan et al. [10] gave a simple algorithm for finding all safe paths, as follows. We can find all safe subpaths in each s-t path $P = (e_1, \ldots, e_{|P|})$ of an arbitrary flow decomposition, since safe paths are part of *all* flow decompositions, using a two-pointer algorithm. Starting with $\ell = 1$ and $r = 2$ as an inclusive interval (i.e., including two edges), we increase r as long as $f_{P[\ell..r]} > 0$, and increase ℓ as long as $f_{P[\ell..r]} \leq 0$. That is, for every index $r = 2, \ldots, |P|$, we find the minimum $\ell < r$ such that the subpath $P[\ell..r]$ is safe. The runtime of this procedure is $O(\text{out-degree}(P))$, where $\text{out-degree}(P) := \sum_{(u,v)\in P} \text{out-degree}(u)$, since flows of edges across P are at most added and subtracted twice throughout the algorithm. To find the set $\mathcal{S}$ of all safe paths of G, we can find any flow decomposition $\mathcal{P}$ and run the two-pointer algorithm on all paths in $\mathcal{P}$. For quickly obtaining a flow decomposition, one could e.g. use the greedy approach of removing paths of largest currently available flow in runtime $O(|\mathcal{P}| \cdot (n + m))$ and use it as upper bound to the optimal solution in the ILP [24]. In our implementations for the three problems MFD, MIFD, MFDSC we use flow decompositons obtained from the fastest heuristic solvers for these problems.

An *(edge) antichain* is a subset $Q \subseteq E$ such that every pair of edges $e_1, e_2 \in Q$ is independent. The size of any edge antichain is also a lower bound on the size of an MFD, as noted by [17], since each edge in the antichain must be traversed by *different* paths in any flow decomposition by the definition of independent edges. The idea is to use such edges (and extensions of them via safe paths, as we discuss next) to fix some ILP variables.



(a) A flow network.

(b) All maximal safe paths with their respective lengths.

(c) Maximum Weight Antichain.

(d) Long collection of independent safe paths.

Figure 2 Application of safe paths in introducing new constraints into previous ILP models. A flow network is displayed in Figure 2a. From this network, all maximal safe paths (that cannot be extended left or right) can be calculated (displayed in different colours with their respective lengths in Figure 2b). We attach to every edge the maximum length of all safe paths crossing that edge as weight. By calculating the maximum weight antichain (Figure 2c), we obtain a set of pairwise independent safe paths (Figure 2d). In the ILP formulation, we set to 1 the x_{uvi} variables of each edge (u, v) in the i-th safe path of the maximum weight antichain.

For every edge $e \in E$, denote by $a(e)$ the length of the longest safe path traversing e, i.e. $a(e) := \max_{P\in\mathcal{S}, e\in P} |P|$; this can be obtained from the computation of safe paths described above. Let $\mathcal{P}(Q) = \{P_1, \ldots, P_{|Q|}\}$ be the corresponding longest safe paths passing through the edges in Q; see Figure 2 for an illustration. If $Q \subseteq E$ is an antichain, then also

the corresponding longest safe paths for the edges of Q must be fully traversed by *different* paths in any flow decomposition (otherwise, if some P_i and P_j were traversed by the same s-t path in some flow decomposition, then also the corresponding edges in Q could be traversed by the same path, contradicting the fact that Q is an antichain).

As such, without loss of generality, we can fix in the ILP the i-th path of the MFD to contain the i-th safe path in $\mathcal{P}(Q)$:

$$x_{uvi} = 1, \quad \forall (u,v) \in P_i, \forall P_i \in \mathcal{P}(Q). \tag{6}$$

Additionally, in order to further optimize the choice of the antichain Q, we consider the following notion. Given a weight function $a : E \to \mathbb{N}$, a *maximum weight antichain* is an antichain Q maximizing $\sum_{e \in Q} a(e)$ [12]. Maximum weight antichains can be found by a reduction to minimum flows, with demands on the edges given by their weights, in runtime $O(m \cdot \sum_{e \in E} a(e)) \subseteq O(n \cdot m^2)$, followed by a depth-first search through the graph. Maximum weight antichains are cut-sets with edges whose minimum flow is equal to their demand [12]. In order to heuristically fix as many x_{uvi} variables as possible, we set the weight $a(e)$ of each edge e to be the length of the longest safe path traversing e. The overall runtime of this preprocessing is thus $O(m \cdot \sum_{e \in E} a(e))$. Note that this approach does not yield the maximum total length of independent safe paths, we describe an approach on how to find them in Appendix B.

As further optimization, note that symmetries are well known to slow down ILP solvers. In the previous ILP formulations, the paths can be arbitrarily permuted, which we can mitigate with the following constraints: $w_{i+1} \leq w_i, \forall i \in \{|Q|+1, \ldots, k-1\}$. We show the complete optimized ILP in Appendix A.2.

Moreover, running the heuristic solvers on the instances beforehand to calculate the safe paths gives us an upper bound to the optimum solution. In our optimized ILPs we have an additional check on whether the lower bound $|Q|$ equals this upperbound. If this holds, then the heuristic flow decomposition is of minimum size, and we directly report it without running the ILP. Thus, safe paths give us the advantage, that they either yield a good lower bound, showing that heuristic methods solved the instance optimally, or can be used to accelerate the ILP when the lower bound is smaller than the solution of the heuristic solution.

▶ Remark 2. Since any flow decomposition satisfying the subpath constraints is (trivially) a flow decomposition, it follows that safe paths for all flow decompositions are also safe for flow decompositions with subpath constraints. However, they do not capture *all* the safe paths for MFDSC. Nevertheless, we can still use such paths with positive excess flow as described above to again reduce the search space of the ILP solver for MFDSC.

▶ Remark 3. Since in the MIFD problem we are given intervals, instead of a single value, for every edge, the excess flow characterization needs to be adapted. In fact, it is an open problem how to find all the safe paths for all feasible inexact flow decompositions (of any size). However, in this paper we consider a "conservative" adaptation of the excess flow notion (*inexact excess flow*), and prove that paths having positive inexact flow are safe for all inexact flow decompositions. As such, we can use them in the same manner as described above to reduce the search space of the ILP solver for MIFD.

▶ **Lemma 4** (Inexact excess flow). *For a path $P = (e_1, \ldots, e_{|P|})$ with $e_i = (u_i, u_{i+1})$ define its* inexact excess flow *as*

$$f_P \coloneqq L_{e_1} - \sum_{\substack{(u_i,v) \in E \\ i \in \{2,\ldots,|P|\}, v \neq u_{i+1}}} R_{u_i v}.$$

A path P is safe if $f_P > 0$.

Proof. Let $f_P > 0$ and assume that no path of a flow decomposition $\mathcal{P}$ of assigned weights $w \in \mathbb{Z}^{+|\mathcal{P}|}$ routed through edge e_1 passes through all of P. This means they all leave P through edges $\text{OUT} := \{(u_i, v) \in E \mid i \in \{2, \ldots, k\}, v \neq u_{i+1}\}$ carrying weight at most $\sum_{e \in \text{OUT}} R_e < L_{e_1}$. This contradicts the assumption that $\sum_{P_j \in \mathcal{P}: P_j(e_1)=1} w_j \in [L_{e_1}, R_{e_1}]$. ◀

Note that we lose the *if and only if* property by assuming the worst case of only sending the lowest possible amount of flow through the first edge and of removing the largest possible amount of flow through the outgoing edges of the path. This means that inexact excess flows will not necessarily find all safe paths.

4 Y-to-V reduction for subpath constraints and inexact flows

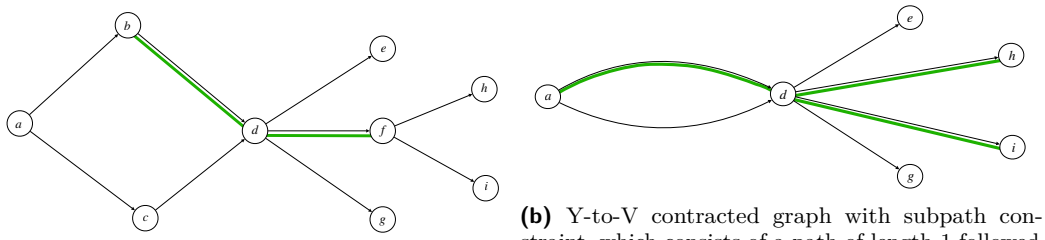
Koster et al. [11, Lemma 4.1] used optimization to reduce the graph size in a pre-processing step by suitably contracting all edges entering nodes with in-degree 1, and respectively, exiting from nodes with out-degree 1, as we review below. This reduction has sometimes been used, under the name "Y-to-V", to simplify the input graphs to other problems, see e.g. [23]. Note that the ILP solver by Dias et al. [4] did not include pre-processing step. In this section, we show how to extent this pre-processing step for the two problem variants MIFD and MFDSC.

The Y-to-V reduction for MFD works the following way. Consider the graph (V, E) and let $v \in V$ be a node of in-degree 1, and let $u \in V$ be the unique node with $(u, v) \in E$. Let v have out-degree ℓ and let $w_i \in V$, $i = 1, \ldots, \ell$ be all nodes with $(v, w_i) \in E$. The reduction creates a new graph $(V \setminus \{v\}, E')$ with edges (u, w_i) and flow $f' : E' \to \mathbb{N}$ with $f'(u, w_i) = f(v, w_i)$ for $i = 1, \ldots, \ell$. This process can be repeated until no nodes of in-degree 1 exist anymore, while all other edges are copied to the new graph. Analogously this can be done for nodes of out-degree 1.

For the MFDSC problem (with subpath constraints), subpaths are given with respect to the original graph, and hence we need to modify them for the Y-to-V contracted graph. The induced subgraphs of nodes of in-degree (resp. out-degree) 1 define a forest of out-trees (resp. in-trees). Subpaths can potentially pass through, begin in or end in such trees (see Figure 3 for an example).

If a path completely passes through a tree, we can merely merge the edges of the tree in the path to the contracted edge. In that case, the subpath which intersects with the tree gets translated a single new edge. If a path stops inside an out-tree or begins inside an in-tree, we can not do that because we do not know which leaf the path will cross in the decomposition. In that case, the subpath which intersects with the tree gets translated to several new, parallel edges.

We first contract all out-trees and describe how to reduce the path to the contracted graph. We can follow up by contracting all in-trees analogously. We can partition the graph uniquely in maximal-size out-trees, such that no adjacent out-trees can be merged to a new out-tree. An out-tree in this partition consists of a single edge if and only if this edge will not be contracted in the Y-to-V reduction. Let the subpath $R_j \in \mathcal{R}$ intersect the out-trees $T_1, \ldots, T_{\ell_j}$, ending at node $u \in T_{\ell_j}$. We can contract the path to obtain R'_j in the following way. Let the set of leaf nodes of T_{ℓ_j} that can be reached from u be denoted by $L(T_{\ell_j}, u)$. For every tree $T_1, \ldots, T_{\ell_j - 1}$, we add an edge from the root of the tree to the leaf the path is crossing to R'_j. For T_{ℓ_j} we add the parallel edges $(\text{root}(T_{\ell_j}), v)$ for all $v \in L(T_{\ell_j}, u)$. That means R'_j is a path of length $\ell_j - 1$ followed by some parallel edges. Let the obtained set of "subpaths" R'_j be $\mathcal{R}'$. We change the constraints (4) and (5) to:



(a) Original graph with one subpath constraint $((b, d), (d, f))$ in green. Every path passing the subpath continues to either node h or node i.

(b) Y-to-V contracted graph with subpath constraint, which consists of a path of length 1 followed by two edges in the last out-tree. Constraint (7) with $\ell_j = 2$ enforces that exactly one of the edges will be used.

Figure 3 A DAG with subpath constraint in green and three out-trees, which are the induced subgraphs of $\{a, b, d\}$, $\{a, c, d\}$ and $\{d, e, f, g, h, i\}$. The subpath constraint passes two out-trees in (a), and is extended to start in an out-tree root to two out-tree leaf nodes in (b).

$$\sum_{(u,v)\in R'_j} x_{uvi} \geq |\ell_j| r_{ij}, \quad \forall i \in \{1, \ldots, k\}, \forall R'_j \in \mathcal{R}', \tag{7}$$

$$\sum_{i\in\{1,\ldots,k\}} r_{ij} \geq 1, \quad \forall R'_j \in \mathcal{R}'. \tag{8}$$

Note that only the set $\mathcal{R}$ and the integers $|R_j|$ change compared to the constraints (4) and (5).

▶ **Lemma 5.** *The constraints* (7) *and* (8) *are true in the original graph if and only if the constraints* (4) *and* (5) *are true in the Y-to-V contracted graph.*

Proof. We analyze the possible routes a path R_j can be extended to, such that it begins and stops at nodes that lie in between different out-trees of the graph partition.

The path R_j begins in the out-tree T_1, potentially not at the root. Since T_1 is an out-tree, it can uniquely be extended to the root w.l.o.g., and we can assume that it starts at the root of T_1.

As such, the path passes through the out-trees $T_1, \ldots, T_{\ell_j - 1}$ from the out-tree root to one of its leaves, which will be the root of the next tree. For these trees, there exists a unqiue edge in the contracted graph, that connects the root to that leaf.

We now analyze the path reduction for the last out-tree T_ℓ. R'_j has been constructed to contain $\ell_j - 1$ edges that form a path in the Y-to-V contracted graph, followed by parallel edges adjacent to that path. These parallel edges describe exactly all possibilities for R_j to be extended to the right, such that it ends outside of the out-tree. Since they are parallel, and the variables x_{uvi} are constrained to represent paths, exactly one of the parallel edges is forced to be used by constraint (7). ◀

The flow conservation is a necessary property for the Y-to-V reduction to work, since because of it, a single out-edge carries no further information. All paths of a flow decomposition entering a node of out-degree 1 must continue through this edge, and will together decompose it (and similarly for nodes of in-degree 1). In the case of MIFD, we do not require the input to have any conservation of flow, and unlike in MFD, valid inputs can also be infeasible to solve.

For nodes of in-degree 1 (resp. out-degree 1) we generalize flow conservation to the property of *inexact flow conservation*, which nodes must have in order for its edges to be contracted: $L_{\text{in}} \leq \sum_{(u,v)\in E} L_{uv} \leq \sum_{(u,v)\in E} R_{uv} \leq R_{\text{in}}$, where $[L_{\text{in}}, R_{\text{in}}]$ is the interval

of the single in-edge of u (respectively for the sum of all outgoing edges and the interval $[L_{\text{out}}, R_{\text{out}}]$ of the single out-edge of u). Note that the second inequality is always fulfilled since $L_{uv} \leq R_{uv}$. The Y-to-V reduction for MIFD works the same as the Y-to-V reduction for MFD, but defines intervals $[L'_{uw_i}, R'_{uw_i}]$ (resp. $[L'_{w_iu}, R'_{w_iu}]$) instead of flow values $f'(u, w_i)$ (resp. $f'(w_i, u)$) and is restricted on nodes of in-degree (resp. out-degree) 1 fulfilling the inexact flow conservation.

▶ **Lemma 6.** *The Y-to-V reduction for MIFD is correct.*

Proof. Consider an inexact flow decomposition of the original graph. By uniquely adapting the paths to the contracted graph, they trivially define a feasible solution, as the same set of weighted paths contribute to the same intervals as before.

Consider an inexact flow decomposition of the contracted graph. By uniquely adapting the paths to the original graph, they define a feasible solution, since the inexact flow conservation enforces the sum of the weights of paths passing through an edge with an interval that has been removed in the contracted graph to lie inside this interval. ◀

5 MFD with given weights

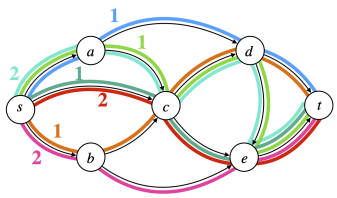


Figure 4 Another flow decomposition of the flow network displayed in Fig. 1a. In this scenario, all weights are already known and limited to only power of 2 (1, 2, 4, 8) values.

In this section we consider an MFD variant in which the path weights in the solution are restricted to belong to a given set $W = \{w_1, \ldots, w_\ell\}$, see also Figure 4 for an example and note that not all weights in W must be used by the solution paths. This problem was initially defined by Kloster et al. [11, Sec. 6], and in this paper we call it *MFD with given weights (MFDW)*. More formally, given a flow network $G = (V, E, f)$ and a set of weights $W = \{w_1, \ldots, w_\ell\} \subseteq \mathbb{Z}^+$, find a minimum-sized set of s-t paths $\mathcal{P} = (P_1, \ldots, P_k)$ with associated weights $(\tilde{w}_1, \ldots, \tilde{w}_k)$, with each $\tilde{w}_i \in W$, such that $\sum_{i \in \{1,\ldots,k\}} P_i(u, v)\tilde{w}_i = f_{uv}$, $\forall (u, v) \in E$.

This problem has a smaller search space than MFD because the weights are restricted to W, and thus can potentially admit faster solvers. Moreover, if the weights of an optimal solution are already known, then an optimal solution to MFDW is also an optimal solution to MFD, and this problem can be used as potentially faster solver. For instance, Kloster et al. [11] have observed that path weights can be found in the flow in graphs used for RNA sequence reconstruction.

In the rest of this section we show that knowing such a set W, we are able to formulate a model with substantially fewer variables than the ILP model for MFD, potentially decreasing the runtime of the ILP solver by a substantial amount. This formulation also has the potential to be an alternative efficient heuristic algorithm for MFD, if W is well chosen.

Given the set of possible weights, our ILP model will be able to answer the following question: In an optimal solution (that minimizes the number of used paths), how many paths of weight w_i pass through an edge e? In other words, to decompose a flow f using the weights in W, we aim to find flows X_i such that $f = \sum_{i=1}^{|W|} w_i X_i$, where each $X_i(e)$ answers that question. The ILP model for MFDW is the following:

$$\textit{Minimize} \quad k := \sum_{(s,u)\in E} \sum_{i=1}^{|W|} x_{sui}$$

Subject to:

$$f_{uv} = \sum_{i=1}^{|W|} w_i x_{uvi}, \qquad \forall (u,v) \in E, \tag{9a}$$

$$\sum_{(u,v)\in E} x_{uvi} - \sum_{(v,u)\in E} x_{vui} = 0, \qquad \forall v \in V \setminus \{s,t\}, \forall i \in \{1,\dots,|W|\}, \tag{9b}$$

$$x_{uvi} \in \mathbb{N}, \qquad \forall (u,v) \in E, \forall i \in \{1,\dots,|W|\}. \tag{9c}$$

Note that the flow $X_i(u,v) = x_{uvi}$ can be decomposed trivially into weight 1 paths. In addition, the product $x_{uvi}w_i$ does not require linearization, due to w_i being an input in this problem.

▶ **Lemma 7.** *The ILP model described by constraints* (9a) *to* (9c) *solves MFDW optimally.*

Proof. Let $\mathcal{P} = \{P_1, \dots, P_k\}$ with weights in $W = \{w_1, \dots, w_\ell\}$ be a flow decomposition. Let $\mathcal{P}(w_i) \subseteq \mathcal{P}$ denote the multiset of paths of weight $w_i \in W$ and let $x_{uvi} = |\mathcal{P}(w_i) \cap \{s\text{-}t \text{ paths crossing } (u,v)\}|$. The variables x_{uvi} define a feasible solution to the ILP model and we have $\sum_{(s,u)\in E} \sum_{i=1}^{|W|} x_{sui} = k$ for the objective function.

Given a feasible solution of the ILP model consisting of variables x_{uvi}, construct s-t paths P of weight $w_i \in W$ in the following way. Start at node s, choose an arbitrary neighbour u of s with $x_{sui} > 0$ and append the edge (s,u) to P. Reduce x_{sui} by one, and continue this step with u instead of s until the current node is t, which finishes the path P. Repeating this process until all the x_{uvi} are zero yields a flow decomposition using $\sum_{(s,u)\in E} \sum_{i=1}^{|W|} x_{sui}$ many paths. ◀

We now further argue that MFDW is strongly NP-hard and that solving MFDW of G with weights $\{2^i \mid i = 0, \dots, \lceil \log ||f|| \rceil\}$ optimally is a $\lceil \log ||f|| \rceil + 1$-approximation of MFD. This is in particular useful when no small set W of optimal solution path weights is known. Depending on the application, weight sets other than powers-of-two can give a better approximation or faster runtime. We can for example use powers of 2^i for any $i \in \mathbb{Z}^+$, which yields an approximation scheme with approximation factor $(2^i - 1)\lceil (\log ||f|| + 1)/i \rceil$. Increasing i, we expect a decreased runtime due to the reduced dimension of the ILP model, but the size of the decomposition increases.

▶ **Lemma 8.** *Let $W = \{w_1, \dots, w_\ell\}$ be a set of weights, and $\mathcal{P}$ with associated weights $W' = \{w'_1, \dots, w'_k\}$ be an optimal solution of an MFD instance $G = (V, E, f)$. If $W' \subseteq W$, then solving MFDW with weight set W optimally solves MFD optimally.*

Proof. This follows from the fact that an optimal solution of MFDW always contains at least as many paths as an optimal solution of MFD. ◀

▶ **Corollary 9.** *MFDW is strongly NP-hard.*

Proof. Hartman et al. have shown that MFD is NP-hard on graphs using flow values only from $\{1, 2, 4\}$ [9]. MFDW with $W = \{1, \ldots, c\}$ solves MFD on instances where the flows are upper bounded by a constant c by Lemma 8. ◀

▶ **Lemma 10.** *For an MFD instance $G = (V, E, f)$, solving MFDW of G with weights $\{2^j \mid j = 0, \ldots, \lceil \log ||f|| \rceil\}$ optimally is a $\lceil \log ||f|| \rceil + 1$-approximation of MFD.*

More generally, solving MFDW of G with weights $\{(2^i)^j \mid j = 0, \ldots, \lceil \log ||f||/i \rceil$ for some $i \in \mathbb{Z}^+$ is a $(2^i - 1)\lceil (\log ||f|| + 1)/i \rceil$-approximation of MFD.

Proof. Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ with associated weights $\{w_1, \ldots, w_k\}$ be an optimal solution to G for MFD and let $\mathcal{P}_w$ be an optimal solution for MFDW. We can construct a feasible solution for MFDW using $\mathcal{P}$ by copying every $P_i \in \mathcal{P}$ for every positive j-th bit in the power of two decomposition of w_i and assigning weight 2^j to that copy. Since $w_i \leq ||f||$, this yields at most $\lceil \log ||f|| \rceil + 1$ paths for every $P_i \in \mathcal{P}$. Thus, $|\mathcal{P}_w| \leq k \lceil \log ||f|| + 1 \rceil$.

If we only use every i-th bit, the bits on positions divisible by i need to cover the next $i - 1$ bits. Let the path P have weight $w = \sum_{j=0,\ldots,\lceil \log ||f|| \rceil} b_j 2^j$ for $b_j \in \{0, 1\}$, then $w_i = \sum_{j=0,\ldots,\lceil \log ||f||/i \rceil} 2^{ij}(b_{ij} + 2b_{ij+1} + \cdots + 2^{i-1}b_{ij+i-1}) = 2^0(b_0 + 2b_1 + \cdots + 2^{i-1}b_{i-1}) + 2^i(b_i + 2b_{i+1} + \cdots + 2^{i-1}b_{2i-1}) + \ldots$. The sums $(b_{ij} + 2b_{ij+1} + \cdots + 2^{i-1}b_{ij+i-1}) \leq 2^i - 1$ denote how often we need to copy P using weight 2^{ij}. ◀

6 Experimental results

Solvers and datasets. Our implementation of the ILPs uses the GUROBI Python API under default settings and is available at `https://github.com/algbio/optimized-fd`. We compare the ILPs for MFD, MFDSC and MIFD also to the heuristic algorithms for them by [20] (Catfish), by [26] (CoasterHeuristic) and by [25] (IFDSolver), respectively. For MFDW we used the weight set $W = \{2^i \mid i \leq \log ||f||\} \cup \{f(e) \mid e \in E\}$.

We also experimented with the implementation from the Catfish solver of the standard greedy algorithm for MFD [24], but since we observed that it performs worse than the Catfish heuristic algorithm, we did not include it in the results. We also did not include in the results the Toboggan implementation by Kloster et al. [11] of the FPT algorithm for MFD, nor the Coaster implementation by Williams et al. [26] of the FPT algorithm for MFDSC since it was already observed in [4] that they do not scale for minimum flow decomposition sizes larger than 6 (which we also confirmed experimentally).

The runtimes of all ILPs (except MFDW) include a linear scan in increasing order to find the smallest k for which there is a flow decomposition in k paths. As discussed in Section 3, the size of the maximum weight antichain Q is a lower bound on k, thus the linear scans for MFD-optimized, MFDSC-optimized and MIFD-optimized start at $|Q|$. We set up a time limit of up to 30 minutes for each input and each method. Our experiments were performed in a server running Linux with one AMD Ryzen Threadripper PRO 3975WX 32-core CPU with 512 GB RAM.

For the MFD problem, we experimented with the datasets created by Shao and Kingsford [20]. These contain graphs created from human transcriptome using the quantification tool Salmon [15], and also datasets created from human and mouse transcriptomes using Flux-Simulator [7]. First, we took a single sample file from their dataset, referred to as *SRR020730 Salmon* (corresponding to the file `rnaseq/salmon/sparse_quant_SRR020730.graph`), followed by the entire archive present in the directories `rnaseq/human/` and `rnaseq/mouse/`, referred to as *Catfish Human* and *Catfish Mouse*, respectively. From those two directories, only the graphs that had 50 nodes

or more (prior to Y-to-V) were considered. For MFDSC, two datasets were used: an adaptation of the *SRR020730 Salmon*, where subpaths were generated and another dataset *SRR30790 StringTie*, which was created by Khan et al. [10] from human RNA-seq reads SRR307903 assembled using the StringTie tool [16]). In both datasets, instances are limited to four subpaths (due to performance limitations regarding previous tools used as benchmarks). Finally, for MIFD, we simulate interval flows similar to what was done in [25]: since infeasible instances are generated in the process, we repeat the process until only feasible instances are generated.

Experimental setup. To show the behaviour of the solvers for graphs of increasingly large MFD, we group the input graphs in ranges based on their MFD size, computed with MFD-optimized. If MFD-optimized does not finish within the time limit on an instance, we exclude it also from all other solvers – note that only the heuristic solvers are faster than MFD-optimized, which are not optimal in general. For each solver, we report the average runtime per range of graphs (column **Avg.**), and the total runtime in that range on the graphs (column **Total**), both in seconds, and only for the graphs on which that solver finished within the time limit. Thus, these numbers are an underestimation of the time one would need to run the solver in practice, since it would run for at least 30 minutes on unsolved instances. In column **#Solved** we list the number of instances on which the solver finished. We captured the runtime with the GNU `time` tool by separating the graphs into individual files and running the tools separately on each instance.

Column $\boldsymbol{\Delta(|\mathcal{P}|)}$ shows the approximation accuracy of the heuristic methods as follows: for each instance, we compute the difference between the number of paths reported by each formulation and the minimum number of paths in an MFD (computed with MFD-optimized). In each table cell, corresponding to a specific range of inputs, we list the sum of these differences, and in parentheses their averages.

Discussion. Table 1 illustrates the performance of the MFD solvers. MFD-optimized improves over MFD-original in all ranges of MFD size. Generally, the larger the MFD size, the more significant the runtime improvement. For example, for Catfish Human ($\min k \geq 16$), MFD-optimized is 27-31× faster than MFD-original on average and for Catfish Mouse ($\min k \geq 21$), MFD-optimized is 23× faster than MFD-original on average.

As mentioned above, in the reported runtimes of the solvers we are not including instances which did not finish within the time limit. Thus, in practice MFD-optimized has even larger speedups compared to MFD-original, since MFD-optimized solves more instances within the time limit than MFD-original, for Catfish Human and Mouse this happened for $\min k \geq 6$. In SRR020730 Salmon and Catfish Mouse MFD-optimized solves all instances, and on Catfish Human MFD-optimized solves 23 more instances.

Among the two heuristic solvers, Catfish finishes on all instances in a few seconds in total, whereas MFDW is slower than Catfish, but still running in under one second on average per graph and faster than MFD-optimized in almost all cases. When comparing the approximation accuracy, note that the instances of practical interest are those for which the MFD size is more than 10, since for smaller sizes MFD-optimized already returns an optimal solution in under 2 seconds on average. On these, MFDW is more accurate than Catfish, and interestingly, the larger the MFD size, the more accurate MFD-optimized becomes (for $\min k \geq 21$ being fully accurate on all datasets). One reason might be that in more complex graphs, the optimal MFD weights appear among the flow values of the edges (which are added to W).

Table 1 Results for Problem MFD. MFD-original denotes the original ILP for MFD from Dias et al. [4] (Section 2), MFD-optimized denotes our optimized ILP described in Section 3 and Section 4, and MFDW the ILP from Section 5. Runtimes are in seconds; a timeout of 30 minutes was used. The total number of instances in the datasets is mentioned in the rows "All", in parentheses. Since the heuristic solvers finish on all instances, we do not have #Solver columns for them.

		Catfish heuristic			MFDW heuristic			MFD-original			MFD-optimized		
	min k	Avg.	Total	$\Delta(\lvert\mathcal{P}\rvert)$	Avg.	Total	$\Delta(\lvert\mathcal{P}\rvert)$	Avg.	Total	#Solved	Avg.	Total	#Solved
SRR020730 Salmon	1-5	0.01	382.38	1 (0.00)	0.29	11150.07	13 (0.00)	0.30	11751.06	38703	0.29	11064.77	38703
	6-10	0.01	19.80	14 (0.01)	0.32	628.14	2 (0.00)	0.48	958.88	1988	0.31	608.25	1988
	11-15	0.01	1.62	6 (0.04)	0.33	53.29	0 (0.00)	2.02	327.71	162	0.34	55.63	162
	16-20	0.01	0.13	0 (0.00)	0.37	4.84	0 (0.00)	15.97	191.68	12	1.27	16.48	13
	21-*max*	0.01	0.06	6 (1.50)	0.40	1.58	0 (0.00)	17.56	52.68	3	0.57	2.28	4
	All (40870)	0.01	403.99	27 (1.55)	0.29	11837.92	15 (0.00)	0.32	13282.01	40868	0.29	11747.41	40870
Catfish Human	1-5	0.00	1.07	3 (0.00)	0.30	3084.70	299 (0.03)	0.35	3631.62	10301	0.29	3010.90	10301
	6-10	0.00	2.42	61 (0.06)	0.35	375.06	243 (0.22)	3.67	3976.98	1083	1.06	1155.47	1085
	11-15	0.01	1.33	53 (0.25)	0.42	89.77	29 (0.14)	16.94	3558.35	210	9.97	2133.17	214
	16-20	0.01	1.23	132 (1.08)	0.39	47.34	6 (0.05)	43.56	4747.77	109	1.26	154.02	122
	21-*max*	0.01	0.10	16 (2.67)	0.43	2.61	0 (0.00)	50.81	101.63	2	1.62	9.69	6
	All (11730)	0.00	6.15	265 (0.02)	0.31	3599.48	577 (0.05)	1.37	16016.35	11705	0.55	6463.25	11728
Catfish Mouse	1-5	0.00	0.94	2 (0.00)	0.30	3578.96	280 (0.02)	0.35	4202.92	12047	0.29	3513.08	12047
	6-10	0.00	6.29	139 (0.09)	0.35	563.83	577 (0.36)	3.29	5247.45	1594	1.07	1706.72	1597
	11-15	0.01	3.16	259 (0.78)	0.55	180.95	173 (0.52)	72.22	21234.09	294	20.84	6877.37	330
	16-20	0.01	0.74	69 (0.93)	0.42	31.26	6 (0.08)	58.18	3607.16	62	6.44	476.43	74
	21-*max*	0.01	0.45	57 (1.84)	0.46	14.28	0 (0.00)	77.59	1862.17	24	3.35	103.75	31
	All (14079)	0.00	11.58	526 (0.04)	0.31	4369.28	1034 (0.07)	2.58	36153.79	14021	0.90	12677.35	14079

In Table 2, we show the results for the MFDSC problem. We observe a similar and even more pronounced trend as for the MFD solvers, where the larger the MFDSC size, the larger the improvement of our optimized ILP. For example, for dataset SRR020730 Salmon, MFDSC-optimized is 11×, 69× and 121× faster than MFDSC-original for min k in ranges 11-15, 16-20, 21-*max*, respectively. For both datasets, MFDSC-optimized manages to solve one more instance than MFDSC-original, and in SRR020730 Salmon it solves all instances. Moreover, the total running time of CoasterHeuristic is more than half of the running time of MFDSC-optimized, while not giving optimal solutions (which are also less accurate on average than the heuristic solvers for MFD).

Table 2 Results for Problem MFDSC. Runtimes are in seconds. A timeout of 30 minutes was used. MFDSC-original is the original ILP from from Dias et al. [4], and MFDSC-optimized is our optimized ILP as described in Section 3 and Section 4. Note that the dataset SRR30790 StringTie has no instances where the optimum MFDSC solution size is larger than 15.

		CoasterHeuristic			MFDSC-original			MFDSC-optimized		
	min k	Avg.	Total	$\Delta(\lvert\mathcal{P}\rvert)$	Avg.	Total	#Solved	Avg.	Total	#Solved
SRR020730 Salmon	1-5	0.19	685.58	109 (0.03)	0.70	2543.97	3643	0.29	1066.78	3643
	6-10	0.19	381.11	411 (0.21)	1.11	2214.29	1988	0.30	598.04	1988
	11-15	0.20	32.39	126 (0.78)	3.79	613.24	162	0.33	53.86	162
	16-20	0.21	2.76	22 (1.69)	70.52	846.29	12	1.01	13.16	13
	21-*max*	0.23	0.90	5 (1.25)	93.28	373.12	4	0.77	3.09	4
	All (5810)	0.19	1102.74	673 (0.12)	1.13	6590.91	5809	0.30	1734.93	5810
SRR30790 StringTie	1-5	0.27	226.04	190 (0.23)	0.69	568.43	828	0.34	280.98	828
	5-10	0.27	63.12	75 (0.32)	1.24	287.75	232	0.37	85.15	232
	11-15	0.24	1.43	1 (0.17)	2.31	11.56	5	0.91	5.44	6
	All (1067)	0.27	290.59	266 (0.25)	0.81	867.74	1065	0.35	371.57	1066

In Table 3, we show the results for MIFD problem and again observe a significant speedup compared to MIFD-original on complex instances. For example, for min k in 11-15, MIFD-optimized is up to 30× faster on SRR020730 Salmon, up to 219× faster on Catfish

Human and up to 100× faster on Catfish Mouse. On the latter two datasets, MIFD-optimized drastically runs in less than 1 second on average. In addition, although IFDSolver has, on average, a better runtime, its difference compared to MIFD-optimized is negligible. Notably, MIFD-optimized solved all instances.

Table 3 Results for Problem MIFD. Runtimes are in seconds. A timeout of 30 minutes was used. MIFD-original is the original ILP from from Dias et al. [4], and MIFD-optimized is our optimized ILP as described in Section 3 and Section 4. Note that the dataset Catfish Human has no instances where the optimum MIFD solution size is larger than 15 and the dataset Catfish Mouse has no instances where the optimum MIFD solution size is larger than 20.

		IFDSolver heuristic			MIFD-original			MIFD-optimized		
	min k	Avg.	Total	$\Delta(\lvert\mathcal{P}\rvert)$	Avg.	Total	#Solved	Avg.	Total	#Solved
SRR020730 Salmon	1-5	0.25	9561.44	2161 (0.06)	0.58	22559.20	38706	0.29	11094.01	38706
	6-10	0.26	512.92	1697 (0.85)	1.03	2044.12	1986	0.32	644.24	1986
	11-15	0.29	46.58	302 (1.88)	11.72	1887.70	161	0.38	60.87	161
	16-20	0.32	4.12	46 (3.54)	28.18	309.97	11	1.25	16.31	13
	21-*max*	0.44	1.74	21 (5.25)	66.30	198.90	3	4.71	18.82	4
	All (40870)	0.25	10126.80	4227 (0.10)	0.66	26999.89	40867	0.29	11834.25	40870
Catfish Human	1-5	0.24	2652.76	957 (0.09)	0.59	6398.51	10911	0.29	3151.02	10911
	6-10	0.26	40.52	198 (1.25)	1.36	215.40	158	0.34	53.57	158
	11-15	0.24	0.98	14 (3.50)	101.01	404.03	4	0.46	1.84	4
	All (11073)	0.24	2694.26	1169 (0.11)	0.63	7017.94	11073	0.29	3206.43	11073
Catfish Mouse	1-5	0.24	3182.89	1521 (0.12)	0.58	7562.22	13009	0.29	3731.19	13010
	6-10	0.25	26.83	139 (1.31)	1.47	155.55	106	0.34	35.90	106
	11-15	0.36	1.79	19 (3.80)	45.35	226.77	5	0.45	2.26	5
	16-20	0.29	0.29	3 (3.00)	0.00	0.00	0	0.56	0.56	1
	All (13122)	0.24	3211.80	1682 (0.13)	0.61	7944.54	13120	0.29	3769.91	13122

Finally, we computed quantitative statistics on the contributions of our various optimizations, see Table 4. Overall, the Y-to-V reduction shrinks the number of edges to less than 50%, even in the inexact flow instances where not every node of in- or outdegree 1 can be contracted. Moreover, the amount of path indicators variables set to 1 by safe paths in an antichain goes up to 21%, and is at least 10% in datasets where less than half of the instances were solved optimally using only heuristic methods (i.e., where the lower bound is equal to the upper bound). The last column shows that the safety predicted about 40% of the final solution paths has been predicted using the independent safe paths.

7 Conclusion

In this paper, we proposed optimizing the ILP formulations for MFD using the notion of safe paths for all flow decompositions. Since safe paths cannot be simply be removed from the graph, we observed that we can use a set of independent safe path to suitably fix a large number of ILP variables corresponding to their edges. Combined with the Y-to-V reduction, and the first usage of an antichain lower bound in a solver to detect the optimality of a heuristic solution, this resulted in a significantly faster MFD solver (up to 34× on the harder instances). We also developed an ILP that can work as a heuristic for MFD, running in under 1 second on average, and more accurate than the state-of-the-art heuristic Catfish on graphs of practical interest. We also showed that these optimizations can be applied to two MFD variants of practical interest (by also adapting the Y-to-V reduction the first time for these variants), resulting in even bigger speedups.

Future research encompasses extending such improvements to the ILPs for MFD in general graphs with cycles [5]. Another point of improvement is a further search space reduction through further optimizations on the application of safety, for example, by answering

Table 4 Quantitative statistics of the various optimisations. The first two columns $|E|$ original and $|E|$ Y-to-V show the number of edges in the original and in the Y-to-V contracted graph (and, in parentheses, the proportion of this number relative to the original graphs), respectively, summed over all graphs in that dataset. The column LB = UB shows the proportion of instances in each dataset where the calculated lower bound (the number of safe paths in a maximum antichain) is equal to the upper bound (the number of heuristically calculated paths). This column indicates in how many instances we have proven that the heuristic solver is optimum using safety. The column p_{IND} shows the proportion of all path indicators x_{uvi} variables that were set to 1 as part of a safe path with respect to all edges of the contracted graph, where i is the index of the path and (u, v) is an edge. The last column $\#\text{IND}/\sum|P|$ shows the proportion of path indicator variables with respect to the sums of the solution's path lengths, since the total solution path length is an upperbound on how many variables we can set to 1.

	$\|E\|$ original	$\|E\|$ Y-to-V	LB = UB	p_{IND}	$\#\text{IND}/\sum\|P\|$
MFD					
SRR020730 Salmon	354519	147996 (41.75%)	32.32%	10.95%	48.60%
Catfish Human	621297	109870 (17.68%)	63.67%	9.03%	41.85%
Catfish Mouse	687763	126388 (18.38%)	54.89%	8.36%	37.37%
MFDSC					
SRR020730 Salmon	214380	94986 (44.31%)	91.45%	8.05%	43.71%
SRR30790 StringTie	24912	8926 (35.83%)	31.02%	15.84%	48.63%
MIFD					
SRR020730 Salmon	355526	166136 (46.73%)	26.77%	11.91%	23.39%
Catfish Human	115774	48903 (42.24%)	34.12%	20.46%	27.29%
Catfish Mouse	101123	40865 (40.41%)	28.05%	21.92%	28.02%

the following question: Is it tractably possible to find subpaths that are part of all α-approximations of MFD, where α can either be a constant or depend on the input size? Such *α-safe paths* (following the notion of c-essential vertices by Bumpus et al. [3]) would be longer the closer α is to 1, while safe paths in this paper would refer to *∞-safe paths*. For the problem variants MFDSC and MIFD, can we find all ∞-safe paths in polynomial time?

References

1. Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *International Conference on Research in Computational Molecular Biology*, pages 221–222. Springer, 2020.
2. Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics*, 30(17):2447–2455, 2014.
3. Benjamin Merlin Bumpus, Bart M. P. Jansen, and Jari J. H. de Kroon. Search-Space Reduction via Essential Vertices. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:15, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.ESA.2022.30`.
4. Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Fast, flexible, and exact minimum flow decompositions via ILP. In *International Conference on Research in Computational Molecular Biology*, pages 230–245. Springer, 2022.

5 Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Minimum flow decomposition in graphs with cycles using integer linear programming. *arXiv preprint arXiv:2209.00042*, 2022.

6 Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20(1):1–14, 2019.

7 Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic rna-seq experiments with the flux simulator. *Nucleic Acids Research*, 40(20):10073–10083, 2012.

8 Michael Hagemann-Jensen, Christoph Ziegenhain, Ping Chen, Daniel Ramsköld, Gert-Jan Hendriks, Anton JM Larsson, Omid R Faridani, and Rickard Sandberg. Single-cell RNA counting at allele and isoform resolution using Smart-seq3. *Nature Biotechnology*, 38(6):708–714, 2020.

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

10 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. Safety and Completeness in Flow Decompositions for RNA Assembly. In Itsik Pe'er, editor, *Research in Computational Molecular Biology - 26th Annual International Conference, RECOMB 2022, San Diego, CA, USA, May 22-25, 2022, Proceedings*, volume 13278 of *Lecture Notes in Computer Science*, pages 177–192. Springer, 2022. doi:10.1007/978-3-031-04749-7_11.

11 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical fpt algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.

12 Ralf Möhring. Algorithmic Aspects of Comparability Graphs and Interval Graphs. In Ivan Rival, editor, *Graphs and Order: the role of graphs in the theory of ordered sets and its applications*. D. Reidel Publishing Company, 1984.

13 Rolf H Möhring. Algorithmic aspects of comparability graphs and interval graphs. *Graphs and order: the role of graphs in the theory of ordered sets and its applications*, pages 41–101, 1985.

14 Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity balancing path flow decomposition and routing. In *2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2015.

15 Rob Patro, Geet Duggal, and Carl Kingsford. Salmon: accurate, versatile and ultrafast quantification from RNA-seq data using lightweight-alignment. *BioRxiv*, page 021592, 2015.

16 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature biotechnology*, 33(3):290–295, 2015.

17 Manuel Ariel Caceres Reyes, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Marshall Mumey, Romeo Rizzi, Alexandru Tomescu, and Lucia Williams. Width helps and hinders splitting flows. In *30th Annual European Symposium on Algorithms (ESA 2022)*, 2022.

18 Zhaleh Safikhani, Mehdi Sadeghi, Hamid Pezeshk, and Changiz Eslahchi. SSP: An interval integer linear programming for de novo transcriptome assembly and isoform discovery of RNA-seq reads. *Genomics*, 102(5-6):507–514, 2013.

19 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35(12):1167–1169, 2017.

20 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017.

21 Alexandru I Tomescu, Travis Gagie, Alexandru Popa, Romeo Rizzi, Anna Kuosmanen, and Veli Mäkinen. Explaining a weighted dag with few paths for solving genome-guided multi-assembly. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(6):1345–1354, 2015.

22 Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. In *BMC bioinformatics*, volume 14, pages S15:1–S15:10. Springer, 2013.

23 Alexandru I Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017.

24 B. Vatinlen, F. Chauvet, P. Chrétienne, and P. Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. doi:10.1016/j.ejor.2006.05.043.

25 Lucia Williams, Gillian Reynolds, and Brendan Mumey. RNA Transcript Assembly Using Inexact Flows. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914. IEEE, 2019.

26 Lucia Williams, Alexandru Tomescu, Brendan Marshall Mumey, et al. Flow decomposition with subpath constraints. In *21st International Workshop on Algorithms in Bioinformatics (WABI 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

A ILP formulations

A.1 MFD-original

$$\forall i \in \{1, \ldots, k\}:$$

$$\sum_{(s,v)\in E} x_{svi} = 1 \tag{10a}$$

$$\sum_{(u,t)\in E} x_{uti} = 1 \tag{10b}$$

$$\sum_{(u,v)\in E} x_{uvi} - \sum_{(v,w)\in E} x_{vwi} = 0 \qquad \forall v \in V \setminus \{s,t\}, \tag{10c}$$

$$f_{uv} = \sum_{i\in\{1,\ldots,k\}} \phi_{uvi} \qquad \forall (u,v) \in E, \tag{10d}$$

$$\forall (u,v) \in E, \forall i \in \{1, \ldots, k\}:$$

$$\phi_{uvi} \leq f_{\max} x_{uvi} \tag{10e}$$

$$\phi_{uvi} \leq w_i \tag{10f}$$

$$\phi_{uvi} \geq w_i - (1 - x_{uvi}) f_{\max} \tag{10g}$$

$$w_i \in \mathbb{Z}^+ \qquad \forall i \in \{1, \ldots, k\}, \tag{10h}$$

$$x_{uvi} \in \{0, 1\} \qquad \forall (u,v) \in E, \forall i \in \{1, \ldots, k\}, \tag{10i}$$

$$\phi_{uvi} \in \mathbb{N} \qquad \forall (u,v) \in E, \forall i \in \{1, \ldots, k\}. \tag{10j}$$

A.2 MFD-optimized

Equations (10) together with:

$$x_{uvi} = 1, \qquad \forall (u,v) \in P_i, \forall P_i \in \mathcal{P}(Q) \tag{11a}$$

$$w_{i+1} \leq w_i, \qquad \forall i \in \{|Q|+1, \ldots, k-1\} \tag{11b}$$

A.3 MFDW

$$\textit{Minimize} \quad \sum_{(s,u)\in E} \sum_{i=1}^{|W|} x_{sui}$$

Subject to:

$$f_{uv} = \sum_{i=1}^{|W|} w_i x_{uvi}, \qquad \forall (u,v) \in E \tag{12a}$$

$$\sum_{(u,v)\in E} x_{uvi} - \sum_{(v,u)\in E} x_{vui} = 0, \qquad \forall v \in V \setminus \{s,t\}, \forall i \in \{1,\dots,|W|\} \tag{12b}$$

$$x_{uvi} \in \mathbb{N}, \qquad \forall (u,v) \in E, \forall i \in \{1,\dots,|W|\} \tag{12c}$$

B Maximum length independent safe paths

The approach used in Section 3 to calculate independent safe paths of a flow network (G, f) does not return them of maximum length, but works only as a heuristic. In order to find independent safe paths of maximum length in polynomial time, one can, as in the heuristic approach, reduce to maximum weight antichains, on the following *dependency* graph $\mathcal{D} = (V(\mathcal{D}), E(\mathcal{D}))$: nodes represent safe paths, with given weights of their length. A directed edge (u, v) is added if there exists a path in G which first traverses both safe paths represented by u and v, entering the safe path of u first and then entering the safe path of v.

▶ **Lemma 11.** *$A \subseteq V(\mathcal{D})$ is an independent set in $\mathcal{D}$ if and only if the set of safe paths corresponding to the nodes in A is independent in G, where the total weight of A is the total length of the corresponding safe paths. As a result, a maximum length independent path set of safe paths in G can be calculated in polynomial time $O(|\mathcal{S}|^2 \cdot \sum_{S\in\mathcal{S}} |S|) \subseteq O(|\mathcal{S}|^2 \cdot n \cdot m)$, where $\mathcal{S}$ is the set of all safe paths.*

Proof. The graph $\mathcal{D}$ is a transitive graph, i.e. $(u, v), (v, w) \in E(\mathcal{D})$ implies that $(u, w) \in E(\mathcal{D})$. That is because path dependencies are transitive: If there is a path P_1 traversing safe paths s_1 and s_2, and there is a path traversing safe paths s_2 and s_3, one can construct a path that traverses all of s_1, s_2 and s_3. Thus, independent node sets in $\mathcal{D}$ correspond exactly to independent safe paths in G: Two safe paths are independent if and only if they are not connected by an edge in $\mathcal{D}$. The graph $\mathcal{D}$ can be constructed in $O(|\mathcal{S}|^2 \cdot m)$ time, by performing a graph search for every pair of safe paths to check whether they are independent or not.

It is a classical result ([13]), that finding a maximum weighted independent node set on transitive graphs is solvable in polynomial time by finding a maximum weight node antichain. Since the weights of the nodes are $|S|$ for $S \in \mathcal{S}$, it runs in time $O(|E(\mathcal{D})| \cdot \sum_{S\in\mathcal{S}} |S|) \subseteq O(|\mathcal{S}|^2 \cdot \sum_{S\in\mathcal{S}} |S|)$. ◀