

DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
REPORT A-2025-2

Algorithms and Graph Structures for Splitting Network Flows, in Theory and Practice

Andreas Grigorjew

*Doctoral dissertation, to be presented for public examination with
the permission of the Faculty of Science of the University of
Helsinki in Auditorium A111, Exactum, on February 19th, 2025,
at 13 o'clock.*

UNIVERSITY OF HELSINKI
FINLAND

Supervisor

Alexandru I. Tomescu, University of Helsinki, Finland

Pre-examiners

Eva Rotenberg, Technical University of Denmark, Denmark

Mingfu Shao, Pennsylvania State University, United States of America

Opponent

László A. Végh, University of Bonn, Germany

Custos

Alexandru I. Tomescu, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi

URL: <http://cs.helsinki.fi/>

Telephone: +358 2941 911

Copyright © 2025 Andreas Grigorjew

ISSN 1238-8645 (print)

ISSN 2814-4031 (online)

ISBN 978-952-84-0769-0 (paperback)

ISBN 978-952-84-0770-6 (PDF)

Helsinki 2025

Unigrafia

Algorithms and Graph Structures for Splitting Network Flows, in Theory and Practice

Andreas Grigorjew

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
contact@andreasgrigorjew.de
<https://www.andreasgrigorjew.de/>

PhD Thesis, Series of Publications A, Report A-2025-2
Helsinki, February 2025, 63 + 77 pages
ISSN 1238-8645 (print)
ISSN 2814-4031 (online)
ISBN 978-952-84-0769-0 (paperback)
ISBN 978-952-84-0770-6 (PDF)

Abstract

Flows on directed graphs measure the number of units that travel between nodes in a network, which naturally models the movement of a commodity. Moreover, a common theme in graph theory is the decomposition of complex objects into simpler ones, as this enables us to work separately with each of the simple objects. Although a vast amount of research handles the computation of flows, a relatively small amount of work addresses their decomposition, despite the relevance of both flows and graph decompositions in theory and applications.

In this thesis, we study the Minimum Flow Decomposition (MFD) problem, which is about finding a smallest set of weighted paths whose superposition is equal to an integral input flow. It is a fundamental optimisation problem that is NP-hard and has applications in various fields, such as Bioinformatics, communication networks or transportation planning. We study the problem on directed acyclic graphs (DAGs) for integer flows and consider two angles, a theoretical and a practical one. The DAG width, the fewest number of paths to cover all edges, plays a central role in both angles.

As many optimisation problems are NP-hard and are not feasibly solvable for many practical instances, heuristics are commonly used as a trade-off between optimality and runtime. Despite the active development of heuris-

tics for MFD in the recent years, we have only gained little understanding of the theoretical aspects of the problem. In the first three papers, we analyse heuristics and their relationship to graph structure. We differentiate between two variants: allowing negative path weights and forcing them to be positive. In the former case, we present an approximation algorithm with performance guarantees for all instances. In the latter, we present performance guarantees for graphs that commonly appear in practice, and we analyse the computational complexity based on notions that are connected to the DAG width.

In the fourth paper, we use Integer Linear Programming (ILP) methods to improve the exact and heuristic solving of MFD. We improve the runtime of the current state-of-the-art exact ILP-based solver using pre-computation steps on the instances. Further, we define a new variant of MFD and show that it can be used as approximation to the original MFD problem, whose approximation performance improves compared to the current state-of-the-art heuristic by a large magnitude while still admitting an acceptable runtime on our datasets.

Computing Reviews (2012) Categories and Subject Descriptors:

Theory of computation → Design and analysis of algorithms →
 Approximation algorithms analysis
 Theory of computation → Design and analysis of algorithms →
 Graph algorithms analysis → Network flows
 Mathematics of computing → Discrete mathematics → Graph
 theory → Graph algorithms
 Applied computing → Life and medical sciences → Bioinformatics

General Terms:

Flow networks, Graph algorithms, Approximation algorithms, Integer
 Linear Programming

Additional Key Words and Phrases:

Flow decomposition, Minimum path cover, Minimum flow, Computational
 complexity, Parameterized complexity, Width, Parallel-width

Acknowledgements

First, I would like to thank my supervisor Alexandru I. Tomescu for his positive attitude and enthusiasm. It was a lot of fun to brainstorm and solve problems together. I am grateful for his enormous support and feedback throughout this journey, which helped me grow as a researcher. I also would like to thank Brendan Mumey, Romeo Rizzi and Hajk-Georg Drost for their wisdom, our collaboration and for inviting me to visit their institutes. I am especially grateful to Brendan for the opportunity to spend two months at Montana State University. Furthermore, I would like to thank the pre-examiners of this thesis, Eva Rotenberg and Mingfu Shao, and my opponent, László A. Végh, for their time and their useful comments and questions.

I want to thank the competitive programming community in Erlangen, where I developed my passion for algorithms and graph theory during my masters. In particular, I want to thank Paul Wild, Daniela Novac, Michael Baer and Jonathan Krebs, from all of whom I have learned a lot.

I would like to thank Pirjo Moen for all the help regarding the process and the bureaucracy of my doctoral studies. I also would like to thank all of my co-authors for the fruitful collaboration, and the whole Algorithmic Bioinformatics group at the University of Helsinki for the pleasant work environment, all the lunches, and our chats. It was a joy to share my time at work with you all. I wish to thank the Research Council of Finland and the European Research Council for the funding that supported my work, my travels, research visits, and my participation in the EPIT 2024 spring school.

Lastly, I want to express my gratitude to my friends that I have met in Finland and Montana, for their encouragement and the moments of joy that we shared. I am incredibly thankful for all the memories. To my family, I am deeply grateful for all their support and love. I would not have achieved this without them.

Helsinki, February 2025
Andreas Grigorjew

Original Publications

This dissertation is based on four original publications. The publications referred to as Papers I-IV are attached to the end of this dissertation:

- I. Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, Lucia Williams. **Width Helps and Hinders Splitting Flows.** In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA 2022)*. Volume 244, pp. 31:1-31:14, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
- II. Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, Lucia Williams. **Width Helps and Hinders Splitting Flows.** *ACM Transactions on Algorithms*, 20(2):13, 2024.
- III. Andreas Grigorjew, Wanchote Jiamjitrak, Brendan Mumey, Alexandru I. Tomescu. **Parameterised Approximation and Complexity of Minimum Flow Decompositions.** *arXiv:2409.20278*, 2024.
- IV. Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, Alexandru I. Tomescu. **Accelerating ILP Solvers for Minimum Flow Decompositions Through Search Space and Dimensionality Reductions.** In *Proceedings of the 22nd International Symposium on Experimental Algorithms (SEA 2024)*. Volume 301, pp. 14:1-14:19, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. Best paper award runner-up.

Besides the four papers above, the author of this dissertation co-authored the following paper:

- Andreas Grigorjew, Artur Gynter, Fernando Dias, Benjamin Buchfink, Hajk-Georg Drost, Alexandru I. Tomescu. **Sensitive inference of**

**alignment-safe intervals from biodiverse protein sequence
clusters using EMERALD.** *Genome Biology, Volume 24:168, 2023.*

Contents

1	Introduction	1
1.1	Overview of the thesis	3
1.2	Individual contributions to original papers	5
2	Preliminaries	7
2.1	Graphs	7
2.2	Flow networks and path covers	9
2.3	Flow decompositions	10
3	Acyclic flow network structure	13
3.1	Motivating example: greedy flow decomposition	13
3.2	Flow derived minors	16
3.3	Width-stable graphs	19
3.4	Flow-width	25
4	Approximation and hardness of Minimum Flow Decomposition	27
4.1	Width helps approximating $\text{MCCD}_{\mathbb{Z}}$	27
4.2	Parity fixing approximation for $\text{MFD}_{\mathbb{N}}$	30
4.3	Hardness of MFD	34
4.3.1	Computational complexity of $\text{MFD}_{\mathbb{N}}$	35
4.3.2	Solving the k -Flow Weight Assignment Problem	37
5	ILP optimisations for practical flow decompositions	41
5.1	ILP models of $\text{MFD}_{\mathbb{N}}$ and its variants	42
5.2	Dimensionality and search space reduction	43
5.3	An ILP solver for weighted $\text{MFD}_{\mathbb{N}}$	49
5.4	Experimental results	50
6	Discussion and future directions	57
	References	59

Chapter 1

Introduction

In this thesis we present theoretical and practice oriented work on the Minimum Flow Decomposition (MFD) problem. We theoretically analyse the approximability of MFD and design efficient algorithms to practically solve the problem. MFD is an optimisation problem, where we are given an integral flow network on a directed graph. The task is to decompose the flow into the smallest number of weighted paths, such that for every edge, its flow is equal to the sum of the weights of the paths that cross this edge. Past research has mostly focused on restricting the flow and path weights to positive integers, which is motivated by the applications in various fields, such as Bioinformatics [3, 9, 15, 29, 35], communication networks [19, 25, 37] or transportation planning [26].

Vatinlen et al. [37] showed that the problem is strongly NP-hard (that is, even when the flow weights are upper bounded by a polynomial in the input length, the problem is NP-hard). To cope with this complexity, heuristics are commonly designed for practical applications. Greedy algorithms for MFD are based on the observation that the difference between flows is again a flow. For example, we can choose a weighted path to subtract from the flow and repeat this until the flow is zero. The resulting weighted paths are a flow decomposition. Vatinlen et al. [37] described the framework of *saturating* algorithms, where the weights of the paths are chosen maximally, such that the paths always saturate¹ an edge (i.e., the weight attached to a chosen path is equal to the smallest flow weight on the path). There exist various greedy strategies on which path to choose in every iteration. The most popular strategy is *greedy-weight*², which chooses paths of largest pos-

¹An edge is saturated, if its flow is equal to 0. In the MFD variant of positive path weights, no more paths can then cross the edge.

²In previous work, this algorithm is often referred to as *greedy-width*. We do not use this term to differentiate it from the graph width parameters that we use in this thesis.

sible weight in every iteration. In the *greedy-length* algorithm, one chooses a longest possible path in every iteration. Hartman et al. [19] showed a worst case approximation ratio of $\Omega(\sqrt{m})$ for greedy saturating approaches and extended the greedy algorithms to decompose a $(1 - \varepsilon)$ fraction of the flow using at most $O(\text{OPT}/\varepsilon^2)$ paths, where m is the number of edges and OPT is the optimal solution.

A heuristic solver that follows a different strategy is *catfish* by Shao and Kingsford [33], which is the current state-of-the-art for heuristically solving MFD. Compared to greedy strategies, where the flow alone is the central object, *catfish* also analyses and modifies the graph structure in a heuristic manner. The tool works with an *optimality gap*, which is defined as the difference between an upper bound and the optimal solution, and reduces it by using two steps: merging edges according to certain criteria, and moving subgraphs (i.e., detaching and reattaching them somewhere else), where the move maintains flow decompositions and does not change the optimal solution size.

Approaches for fast, exact solving of MFD have also been studied in previous work. Kloster et al. [22] showed that MFD is fixed parameter tractable by the optimal solution size, and presented an algorithm that runs in time $2^{O(k^2)} \cdot (m + \log \|X\|)$, where k is the optimal solution size, m is the number of edges and $\|X\|$ is the largest flow weight of all edges. Recently, Dias et al. [10] designed an Integer Linear Program (ILP) model for MFD, and showed that solving it is the currently fastest method to compute a minimum flow decomposition and that it can be extended fairly easily in many contexts that are useful for practical applications (for example, in order to mitigate noisy data).

Despite the study of heuristics for MFD in previous research, it lacks results and understanding of the ease and hardness of the approximation of the problem. The performance of heuristics for MFD is usually proven empirically [19, 25, 33]. However, practical instances can have a relatively simple structure. For example, in RNA transcript reconstructions, we use reads (short substrings of a transcript) and their abundances (the number of times a read occurs) to construct a flow network. The nodes represent exons, and edges are created from one exon to another if they appear consecutively in a read. For every edge, we consider the sum of abundances of all reads where the two exons appear consecutively. In optimal conditions, where the coverage of the reads is evenly distributed throughout the whole transcript, this weight assignment is equal to a flow; however, this is not the case in practice due to noise in the data, as reads are usually not evenly distributed, and reads contain errors [33]. The resulting graph is a DAG

(directed acyclic digraph) and is series-parallel, which is a class of graphs that often occurs in practice.

MFD is also used in communication networks, as Hartman et al. [19] described, where data is sent through a network connected through routers. Minimising the number of paths that decompose this network minimises the necessary cost to maintain the routers among other benefits [19]. A large decomposition introduces additional costs that are amplified over time. Despite the NP-hard complexity of the problem, we are thus also interested in expanding the set of instances that can be solved exactly in a feasible amount of time.

Moreover, the problem is interesting for the theory of flow networks itself, as it follows a common pattern in graph theory. We are given a complex object (the flow network) and aim to decompose it into several simple objects (weighted paths), each of which can be understood and worked with more easily. The study of graph decompositions is massive and is an active area of research (see, e.g., [5, 30, 31], among many other works).

This thesis aims to increase our knowledge of MFD approximation and to improve practical MFD solving. We show what makes approximating the optimal solution hard and in which classes of instances it remains simple. This gives a first theoretical explanation for the strong performance of common heuristics. Moreover, we optimise the recently introduced ILP for MFD by Dias et al. [10], using pre-computations to simplify the input for the ILP solver.

1.1 Overview of the thesis

The thesis is split into two parts. In the first part, which makes up around two thirds of the thesis, we theoretically analyse the approximation of MFD. In the second part we improve the state-of-the-art for practical solving MFD, by introducing an ILP based heuristic solver and by optimising the existing ILP model for exact solving. We start by introducing the basic notation and the necessary tools in Chapter 2.

In Chapter 3 we analyse relevant graph structure for MFD. We provide first evidence that the width of a DAG G , which is defined as the smallest number of paths needed to cover all edges of G , plays a vital role in MFD. We construct a class of instances where the width grows exponentially after saturating edges, which improves the currently best known lower bound of the approximation factor of greedy-weight from $\Omega(\sqrt{m})$ [19] to $\Omega(m/\log m)$, where m is the number of edges. We also introduce the class of *width-*

stable DAGs and show that greedy-weight admits an approximation factor of $O(\log \text{Val}(X))$ on it, where $\text{Val}(X)$ is the total flow value of the flow X . Width-stable DAGs are highly relevant for practice and generalise series-parallel DAGs in a natural way. We, moreover, generalise width-stability by showing a close connection of MFD to a notion of directed graph minors defined by Deligkas and Meir [8], which provides us with a larger set of graph structural tools.

In Chapter 4 we consider another approximation strategy and express the input flow as the sum of flows whose decomposition is small and can be performed in polynomial time. We use a parity-fixing approach and show that it admits a $O(\log \|X\|)$ approximation factor for MFD, when we allow negative path weights to appear, where $\|X\|$ is the maximum absolute values of all flow weights. We further show that for the variant restricted to positive path weights, this approach admits the same approximation factor on width-stable DAGs and on instances whose optimal solution size is at least the *parallel-width*, which was defined by Deligkas and Meir [8] in the context of directed graph minors as the largest minimal set of edges that form an s - t cut of the graph. We introduce the width of a flow network $\text{fw}(G, X)$, which connects the DAG width to the parallel-width by generalising both of the notions. Using this connection, we obtain an approximation ratio of $O(\frac{\text{par-width}(G)}{\text{width}(G)} \cdot \log \|X\|)$ for the flow network (G, X) . Moreover, this algorithm can admit an $\Omega(\|X\|)$ approximation factor in the worst case, as we show that the ratio between the parallel-width and the minimum flow decomposition size can grow arbitrarily large. This leads to the remaining class of instances whose $O(\log \|X\|)$ factor approximation is still left open. Complementing the approximation results, we analyse the parameterised complexity of MFD with the width or parallel-width. We show that MFD is strongly NP-hard if the width is constant, and NP-hard if the parallel-width is constant. We also show, as a result of the approximation factor that we achieved, that the computational complexity of MFD with constant parallel-width is quasi-polynomial, if the flow is stored with unary bits.

In Chapter 5 we proceed to optimising the ILP introduced by Dias et al. [10]. Among various optimisations, we utilise the DAG width and solution safety [36], which is a framework that calculates sub-solutions that are present in all (not necessarily optimal) solutions of a problem, in order to reduce the search space of the ILP. We demonstrate how to adjust the optimisations for practical variants that use inexact flows [38], which we model using possible intervals for every edge instead of specific flow weights, and subpath constraints [16, 39], where in addition to the flow, we are given subpaths that must occur in the solution. Furthermore, we define an ILP

model for a new variant, which restricts the path weights to come from a given set, and show that it acts as an efficient heuristic for MFD by guessing the weights. We achieve speedups for the ILP of up to $34\times$ on the standard variant of MFD and a striking factor of up to $219\times$ on an MFD variant. We also show that the ILP model for the weighted MFD variant can compete with the state-of-the-art heuristic Catfish [33], as it is more accurate in complex instances, but it obtains a slower runtime.

1.2 Individual contributions to original papers

Paper I. This paper shows that the width (the minimum number of paths needed to cover all edges) plays an important role in MFD, using it to improve the approximation factor lower bound of greedy-weight, and to define and analyse an approximation algorithm for MFD when negative path weights are allowed. We also disprove a stated conjecture related to a variant of MFD. The individual contributions by the author of this thesis were to polish the proof for the greedy-weight lower bound, to improve the presentation of the approximation algorithm and to formalise the key ideas of the counter example to the conjecture.

Paper II. This paper extends Paper I. It gives a combinatorial classification of width-stable graphs and generalises the approximation algorithm that allows negative integers to graphs with cycles and circulation decompositions. The individual contributions by the author of this paper was to find and prove the classification of the width-stable graphs, to write down the generalisation and to prove that it is correct.

Paper III. This paper analyses a heuristic by Mumey et al. [25] for the variant of MFD where only positive weights are allowed, using a graph parameter. This parameter was first introduced by Deligkas and Meir [8] outside the context of flow networks. This paper uses the parameter and shows a connection of flow decompositions to directed graph minors. The individual contributions of the author of this thesis were to find and prove the connection to graph minors, to prove the improved worst case lower and upper bounds of the heuristic and to design one of the parameterised hardness reductions.

Paper IV. This paper optimises a recent ILP solver by Dias et al. [10] and defines a new ILP model for a weighted variant of MFD. The individual contributions of the author of this thesis were to adapt the optimisations to variants of MFD, to explain the optimisations and the new ILP solver, to

prove their correctness, to implement them, and to experimentally measure their performance.

Chapter 2

Preliminaries

In this chapter, we discuss the concepts necessary for the topics of this thesis. This includes formal definitions and technical notation. In addition, we introduce the first basic results. This chapter can be used as a glossary for the reader throughout the thesis.

2.1 Graphs

We primarily work on directed graphs. We are given a tuple $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a multiset of edges, which are ordered tuples of nodes¹. In this work, we always use the notation V for the nodes and E for the edges when the context towards which graph they refer to is clear. We assume that E is a multiset, which means that we allow multiple occurrences of the same tuple in E . We call such edges parallel. In order to define functions $X : E \rightarrow \mathbb{Y}$ to some set $\mathbb{Y}$, we must be able to identify each parallel edge. We can do so by adding a subscript index i to every parallel edge $(u, v)_i$, where $i \in [k] := \{1, 2, \dots, k\}$ if the edge (u, v) appears k times. Unless it is necessary for clarity, we omit the subscript index. We also say that a node u is connected to a node v if $(u, v) \in E$. We use the notation $n = |V|$ and $m = |E|$.

We say that a graph $G' = (V', E')$ is a subgraph of $G = (V, E)$, written $G' \subseteq G$, if $V' \subseteq V$ and $E' \subseteq E$. We call a sequence of edges $P = (e_1, \dots, e_\ell)$ with $e_i = (u_i, v_i)$ a path in a graph G , if $v_i = u_{i+1}$ for all $i \in [\ell - 1]$. We call P a u_1 - v_ℓ path to indicate the first and last nodes of the path. A cycle is a non-empty v - v path. Two edges (u_1, v_1) and (u_2, v_2) are called independent

¹In some occasions we use undirected graphs, where the edges form a multiset of sets (i.e., unordered tuples) $E \subseteq \{\{u, v\} \mid u, v \in V\}$.

if there exists no v_1 - u_2 path and no v_2 - u_1 path. We call a set of pairwise independent edges an antichain.

In the whole of this thesis, apart from Section 4.1, we assume that all directed graphs are acyclic (DAGs): For each $u \neq v \in V$, there is not both a u - v path and a v - u path in G . We also assume that $(v, v) \notin E$ for all $v \in V$.

For a node $v \in V$, we define $N^+(v) \subseteq V$ (resp. $N^-(v) \subseteq V$) as the multiset of nodes u for every $(v, u)_i \in E$ (resp. $(u, v)_i \in E$). In particular, u is included in $N^+(v)$ (resp. $N^-(v)$) k times if there are k edges $(v, u)_i$ (resp. $(u, v)_i$). We define $\deg^+(v) = |N^+(v)|$ and $\deg^-(v) = |N^-(v)|$ as the out- and in-degree of v , respectively.

On DAGs, there always exists a set of source nodes with in-degree 0 and a set of sink nodes with out-degree 0. We assume that all DAGs have a unique source node s and a unique sink node t , resulting in an s - t DAG. We can transform any DAG to s - t DAGs by adding a new node $s \notin V$, and adding edges (s, v) for all source nodes v , and doing so analogously for t . In Section 4.1, we assume that the directed graphs have no source and no sink node. A cut-set of a DAG G is a set of edges C , such that every s - t path in G crosses C .

Definition 2.1 ([Paper I, Definition 19] Series-parallel graph [12]). An s - t graph is a (*two-terminal*) *series-parallel* graph, if it has two terminal nodes: a single source s , a single sink t , and if:

- G consists of a single directed edge from s and t and has no other nodes, *or*
- it can be obtained from two (smaller) series-parallel graphs G_1 and G_2 , with terminal nodes s_1, t_1 , and s_2, t_2 , respectively, by either
 - identifying $s = s_1 = s_2$ and $t = t_1 = t_2$ (*parallel composition* of G_1 and G_2), *or*
 - identifying $s = s_1$, $t_1 = s_2$, and $t = t_2$ (*series composition* of G_1 and G_2).

A lot of NP-hard optimisation problems become tractable on series-parallel graphs (see, for example, [20, 23]).

Definition 2.2 ([14]). A *funnel* F is an s - t DAG, whose s - t paths all contain a *private* edge: this edge gets traversed by no other s - t path.

Equivalently, Millani et al. [14] also characterised funnels as a generalisation of directed trees. The vertex set of a funnel can be partitioned into two

sets, such that every s - t path first traverses an out-tree T_{out} (i.e., $\deg^-(v) = 1 \leq \deg^+(v)$) and then an in-tree T_{in} (i.e., $\deg^-(v) \geq 1 = \deg^+(v)$). The edges in $E \cap (T_{\text{out}} \times T_{\text{in}})$ are private edges and $E \cap (T_{\text{in}} \times T_{\text{out}}) = \emptyset$. We call a node v in a funnel F *forking*, if $\deg^+(v) > 1$, and *merging*, if $\deg^-(v) > 1$.

2.2 Flow networks and path covers

Flow networks are the main topic in this thesis. Given an s - t DAG G , we call functions $X : E \rightarrow \mathbb{Y}$, where $\mathbb{Y}$ is some integer set, a *flow*, if it satisfies the *flow conservation*:

$$\sum_{u \in N^-(v)} X(u, v) = \sum_{u \in N^+(v)} X(v, u) \quad \forall v \in V \setminus \{s, t\}.$$

In Section 4.1 we use *circulations*, which are functions $X : E \rightarrow \mathbb{Y}$ on graphs which fulfil the flow conservation on all nodes.

We call the tuple (G, X) a flow network, where G is a graph and X is a flow/circulation on G . We denote by $X + Y$ and αX the element wise sum and scalar multiplication of flows/circulations, and remark that they are again flows/circulations. In particular, the zero function 0 on the edges is a flow/circulation. Given two functions $f, g : E \rightarrow \mathbb{Y}$, we write $f \leq g$ (resp. $f < g$) to denote $f(e) \leq g(e)$ (resp. $f(e) < g(e)$) for every $e \in E$. We denote by $\text{Val}(X) = \sum_{u \in N^+(s)} X(s, u)$ the *flow value* of a flow X . For a flow/circulation X , we call the value $X(e)$ for $e \in E$ the flow weight of X on e , we denote by $\text{supp}(X) = \{e \in E \mid X(e) \neq 0\}$ the set of edges with non-zero flow weight and by $\|X\| = \max\{|X(e)| \mid e \in E\}$ the maximum norm of X . We write $X \equiv_2 Y$ for two flows/circulations X, Y , if they have the same parity on every edge.

Let $\ell, u : E \rightarrow \mathbb{Y}$ be given functions. We call the problem of finding a flow X of minimum $\text{Val}(X)$ (resp. maximum $\text{Val}(X)$) such that $\ell \leq X \leq u$ the min-flow (resp. max-flow) problem.

On a DAG, s - t paths P naturally define unit valued flows by their indicator function: $P : E \rightarrow \{0, 1\}$ with $P(e) = 1$ if and only if $e \in P$. By abuse of notation, we use the same variable as for the path itself to denote this function. We further say that a u - v path P on G with flow X carries flow μ , if $X(e) \geq \mu$ for all $e \in P$. We also define that any length 0 u - v path carries flow ∞ .

We call a set of paths $\mathcal{P} = \{P_1, \dots, P_k\}$ in G a path cover of $S \subseteq E$, if $S \subseteq \{e \in E \mid \exists P_i \in \mathcal{P} : e \in P_i\}$. Minimising the cardinality of $\mathcal{P}$ is the well known *Minimum Path (Edge) Cover* problem, which is a variant of the more common *Minimum Path Vertex Cover*, that asks for the minimum

number of paths to cover all nodes of G . By Dilworth's theorem², this number is equal to the number of edges in a maximum size antichain [11].

Definition 2.3 ([Paper I, Definition 6]). Given $S \subseteq E$, we define $\text{width}_S(G)$ as the minimum number of s - t paths in a DAG G needed to cover all edges of S . If $S = E$ we call it the DAG width and write $\text{width}(G)$.

Flows and path covers are equivalent objects in the following sense: given a path cover $\mathcal{P} = \{P_1, \dots, P_k\}$ and a vector $w \in \mathbb{Y}^k$, we can define a flow $X : E \rightarrow \mathbb{Y}$ by $X(e) = \sum_{i=1}^k w_i P_i$. Conversely, given a flow $X : E \rightarrow \mathbb{Y}$, there exists a path cover $\mathcal{P} = \{P_1, \dots, P_k\}$ and a weight vector $w \in \mathbb{Y}^k$, such that $X(e) = \sum_{i=1}^k w_i P_i$. This is known as the *Flow Decomposition Theorem* [1]. For example, if $X \geq 0$, we can iteratively find an s - t path P that traverses non-zero flow edges and subtract it from X . We then continue with the remaining flow $X = X - P$ until it is zero. We call X the induced flow of $\mathcal{P}$ and $\mathcal{P}$ an induced path cover of X , respectively.

As described by, for example, Ahuja et al. [1] or Cáceres et al. [6], we can find a minimum path cover of a given set $S \subseteq E$ by solving a min-flow problem with lower bound constraints $\ell(e) = 1$ for $e \in S$ and $\ell(e) = 0$ for $e \in E \setminus S$, and upper bounds ∞ on every edge. The min-flow problem can be reduced to the max-flow problem, which can be solved in $O(nm)$ time (see, e.g., [27]). We thus have the following lemma.

Lemma 2.4 ([1, 27]). *Let $G = (V, E)$ be a DAG, and $S \subseteq E$. A flow $C : E \rightarrow \mathbb{N}$ can be computed in $O(nm)$ time, such that $C(e) \geq 1$ for all $e \in S$ and $|C| = \text{width}_S(G)$.*

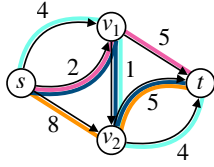
2.3 Flow decompositions

Decomposing objects are a common theme in graph theory [5, 30, 31]. In this thesis, we are interested in decomposing flow networks, which we formulate as the following optimisation problem.

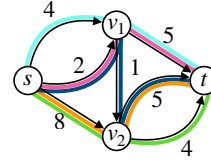
Definition 2.5 ([Papers I-III]). Given a flow X , a *flow decomposition* of X of size k is a family of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ with weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $X = w_1 P_1 + \dots + w_k P_k$.

We denote by $\text{mfd}_{\mathbb{Y}}(G, X)$ the smallest size of a flow decomposition of X with weights in $\mathbb{Y}$, and we call the problem of finding a flow decomposition of size $\text{mfd}_{\mathbb{Y}}(G, X)$ the *Minimum Flow Decomposition* (MFD) problem. We

²Dilworth's theorem can be applied on the partial order on the edges defined by $(u_1, v_1) \leq (u_2, v_2)$ iff there exists a v_1 - u_2 path in G .



(a) If negative weights are allowed, we can decompose the flow in 4 paths with weights 4, 5, 8, and -3 (dark blue).



(b) If negative weights are not allowed, five paths are needed, since the edge (v_1, v_2) must be decomposed by a weight 1 path, leaving 4 edges that must be covered separately.

Figure 2.1: [Paper I, Figure 1] Flow decomposition of a positive flow X . We have $\text{mfd}_{\mathbb{N}}(G, X) = 5$ and $\text{mfd}_{\mathbb{Z}}(G, X) = 4$.

consider the variant $\text{MFD}_{\mathbb{N}}$, where the flow and path weights are assumed to consist of non-negative integers, and the variant $\text{MFD}_{\mathbb{Z}}$, where we also allow negative integers to occur in the flow and in the path weights. See Figure 2.1 for an example.

In $\text{MFD}_{\mathbb{N}}$, we call an edge of flow 0 *saturated*. This terminology is motivated by saturating edges in the max-flow problem: When computing a max-flow Y with capacities u , an edge e is saturated if $Y(e) = u(e)$. If we consider $\text{MFD}_{\mathbb{N}}$ on (G, X) from the bottom-up perspective, then the problem is a max-flow problem with upper bound capacities $u = X$, where we additionally maintain the minimum flow decomposition during the computation of the max-flow. In this thesis, we consider the top-down perspective, that is, we remove flow from the input and maintain a small flow decomposition of the removed flow.

Vatinlen et al. [37] showed bounds $\lceil \frac{m}{n-1} \rceil \leq \text{mfd}_{\mathbb{N}}(G, X) \leq m - n + 2$ for all $X > 0$ and proved that MFD is *strongly* NP-hard. This follows from a reduction from 3-PARTITION that we show in Figure 2.2. This reduction also shows that MFD is strongly NP-hard on series-parallel graphs. However, MFD can be solved in polynomial time in funnels. In fact, any flow X on a funnel has a unique flow decomposition (assuming that the flow decomposition does not repeat paths).

MFD is a relevant problem in practice. For example, $\text{MFD}_{\mathbb{N}}$ is used in Bioinformatics for biological sequence reconstruction [2, 3, 9, 15, 29, 34, 35, 38, 39], in communication networks [19, 25, 37] or in transportation planning [26]. To cope with the complexity of MFD , research has mainly focused on finding heuristics that have been proven to be efficient experimentally on practical datasets [19, 25, 29, 33].

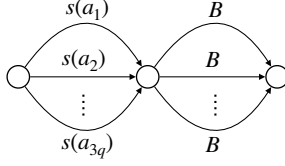


Figure 2.2: [Paper II, Figure 1] The reduction of 3-PARTITION to MFD by Vatinlen et al. [37]. The 3-PARTITION instance consists of a set $A = \{a_1, \dots, a_{3q}\}$, where every a_i has a positive integer size $s(a_i)$, and a positive integer B , such that $B/4 < s(a_i) < B/2$ holds for every $a_i \in A$. The question is whether A can be partitioned into q disjoint subsets, each of 3 elements and of size B . As Vatinlen et al. [37] showed, we obtain a solution to 3-PARTITION if and only if we can decompose the flow in $3q$ paths.

Given a flow network (G, X) , we say that a polynomial algorithm is a factor $f(G, X)$ approximation for MFD, if the decomposition produced of the algorithm consists of at most $f(G, X) \cdot \text{mfd}_{\mathbb{Y}}(G, X)$ paths. Hartman et al. [19] showed that $\text{MFD}_{\mathbb{N}}$ is APX-hard, even when the flow consists of weights in $\{1, 2, 4\}$. This means that there exists a constant $\varepsilon > 0$, such that there is no $(1 + \varepsilon)$ -factor approximation algorithm, unless $\text{P} = \text{NP}$.

Vatinlen et al. [37] described a heuristic framework for solving $\text{MFD}_{\mathbb{N}}$: pick any optimality criteria that chooses *saturating* weighted s - t paths (i.e., a weighted path decomposes at least one edge) and successively remove paths that are optimal under this criteria. Such algorithms are called *greedy saturating* algorithms. The most commonly used example is *greedy-weight*, which chooses paths by the largest possible weight. Other examples include *greedy-long* (used, for example by the DNA assembly tool *vg-flow* [2]) or *greedy-short*, which choose paths by maximising/minimising the lengths and then assign them the largest possible weight. Hartman et al. [19] showed that greedy-weight admits an approximation factor of $\Omega(\sqrt{m})$ in the worst case. We further improve this worst case lower bound in Chapter 3.

MFD is in FPT parameterised by $k := \text{mfd}_{\mathbb{Y}}(G, X)$ and can be solved in time $4^{k^2} k^{1.5k} k^{o(k)} 1.765^k \cdot (n + \log \|X\|) = 2^{O(k^2)} \cdot (n + \log \|X\|)$ [22]. It can also be solved via an Integer Linear Programming (ILP) model [10], which is the currently fastest exact solver. We will improve the runtime further in Chapter 5.

Chapter 3

Acyclic flow network structure

In this chapter, we explore how MFD relates to the structure of DAGs. For a flow network (G, X) with $X > 0$, a natural lower bound to $\text{mfd}_{\mathbb{Y}}(G, X)$ is $\text{width}(G)$, since every flow decomposition of X is a path cover of G . Surprisingly, the width has not previously been considered in the research of MFD.

We first show that the width is relevant by improving the lower bound of the greedy saturating algorithms for $\text{MFD}_{\mathbb{N}}$, where it grows very large after saturating edges. Next, we show that there exists a notion of directed graph minors that is closely related to the decomposition of flows in $\mathbb{N}$. Graph minors were first introduced by Robertson and Seymour [30] in a series of 23 papers, which has initiated the theory of graph structure. Finally, we introduce the class of width-stable graphs, whose width does not increase in the process of the flow decomposition, and show that they naturally generalise the class of series-parallel graphs.

3.1 Motivating example: greedy flow decomposition

We show a class of graphs, for which greedy-weight obtains a large approximation factor lower bound. The graph structural background behind it is the fact that the width of the DAG, as a lower bound to the minimum flow decomposition, can grow exponentially in the process of the decomposition.

We recursively define the $\text{MFD}_{\mathbb{N}}$ instances $(G_{\ell}, X_{\ell, B})$ for which greedy-weight is a $\Omega(\frac{m}{\log m})$ -approximation in Figure 3.1. This improves the previously known lower bound of $\Omega(\sqrt{m})$ [19]. The value B is a positive integer that, for every ℓ , appears as parameter on the flow of edges that form a single s - t path. We call this path the *backbone path*, and we call the edges

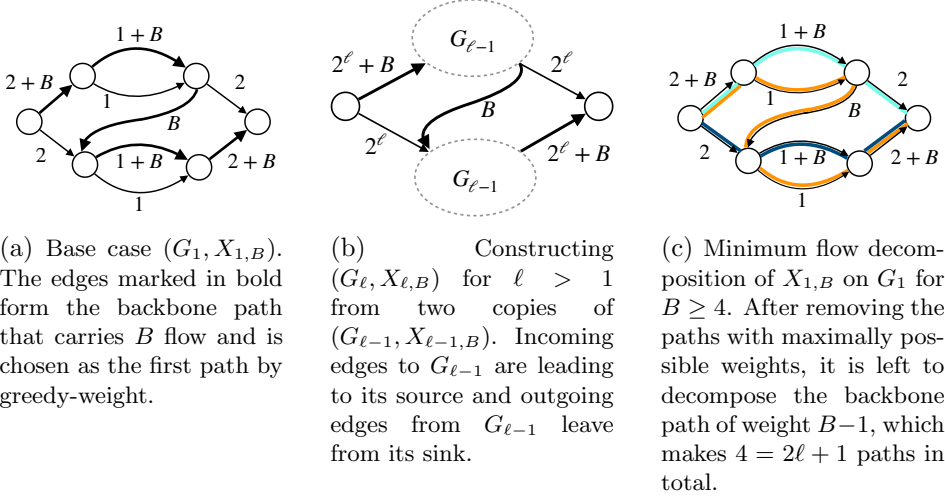


Figure 3.1: [Paper I, Figure 3] Construction of MFD instances $(G_\ell, X_{\ell,B})$, which obtain the exponential gap of a $\Omega(\frac{m}{\log m})$ factor more paths for greedy-weight compared to the optimal solution. Edges whose flow depends on B form a path, which we call the *backbone path*.

e with $X_{\ell,B}(e) = B$ *central edges*. The idea of the construction is to choose B large enough in order to have two effects: 1) the large flow on the central edges enables the minimum flow decomposition to use only $\mathcal{O}(\ell)$ paths and 2) greedy-weight chooses the backbone path as the first path to remove from the flow, and decomposes all and only all of the central edges. Doing so will result in a decomposition size of $\Omega(2^\ell)$.

Remark 3.1. The following facts are easily observed:

1. The flow instances in the construction are well-defined: The flow conservation holds for all internal vertices (all vertices but s, t) in the base case $(G_1, X_{1,B})$. The flow value of $X_{\ell,B}$ is $2^{\ell+1} + B$, and by induction, the flow conservation holds for all internal vertices in the graph $(G_{\ell+1}, X_{\ell+1,B})$.
2. $(G_\ell, X_{\ell,B})$ contains exactly $2^{\ell-1}$ copies of $(G_1, X_{1,B})$ with a cut-set of $2^{\ell+1}$ edges.

Lemma 3.2 ([Paper I, Lemma 12]). *For all $\ell \geq 1$, greedy-weight decomposes the flow $X_{\ell,2^{\ell+1}}$ on G_ℓ into $1 + 2^{\ell+1}$ weighted paths.*

Proof sketch. Since $B = 2^{\ell+1}$, the backbone path carries the largest possible flow, and thus it will be the first weighted path to be decomposed

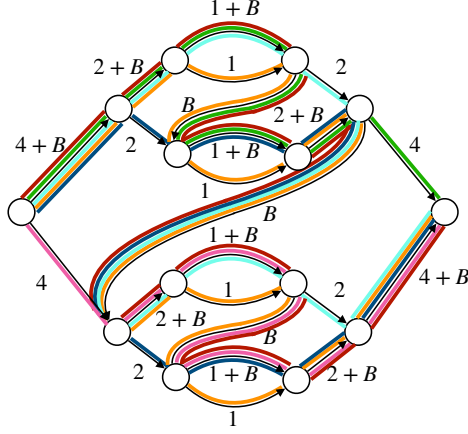


Figure 3.2: [Paper I, Figure 4] A decomposition of $X_{2,B}$ on G_2 with $B \geq 2^3$ into $2 \cdot 2 + 2 = 6$ weighted paths.

by greedy-weight. As a result, all the central edges will be saturated. By Remark 3.1, the graph contains a cut-set of size $2^{\ell+1}$, which becomes an antichain after saturating the central edges. $\square$

Lemma 3.3 ([Paper 1, Lemma 10]). *For all $\ell \geq 1$, the flow $X_{\ell, 2^{\ell+1}}$ on G_ℓ can be decomposed into $2\ell + 2$ weighted paths.*

Proof sketch. We must show that the non-backbone edges can be decomposed into $2\ell + 1$ weighted paths for all instances using induction, using a total weight of $2^{\ell+2} - 3$. The remaining backbone path can then be decomposed in a single path of weight 3.

The decomposition of the base case $\ell = 1$ is shown in Figure 3.1c, in which the orange path saturates both of the edges of flow 1 by traversing the central edge, and the remaining $2 = 2\ell$ paths (in dark blue and light blue) saturate the edges of flow 2. In the flow network remains the backbone path carrying flow $B - 1 = 3$.

We use induction for the remaining step. Assume that the non-backbone edges in $(G_\ell, X_{\ell, 2^{\ell+1}})$ can be decomposed into $2\ell + 1$ weighted paths. We then use this decomposition to decompose the non-backbone edges in the flow network $(G_{\ell+1}, X_{\ell+1, 2^{\ell+2}})$ into $2(\ell + 1) + 1$ weighted paths. See Figure 3.2 for an application of this proof in the case $(G_2, X_{2, 2^{2+1}})$. $\square$

Theorem 3.4 ([Paper I, Theorem 1]). *The approximation ratio of greedy-weight is $\Omega(\frac{m}{\log m})$, in the worst case.*

Proof sketch. First, we can prove by induction that $m = 2^\ell \cdot 7 - 5$ for all $\ell \geq 1$.

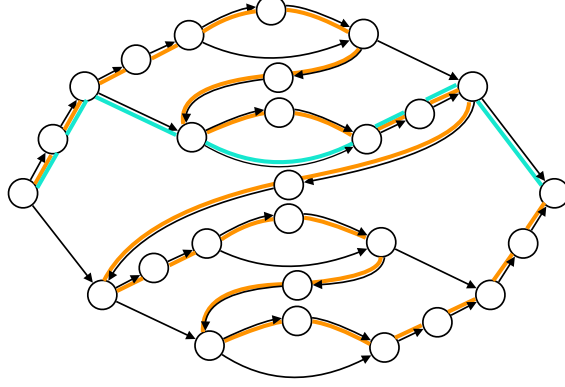


Figure 3.3: [Paper II, Figure 5] By subdividing the backbone edges, the orange path becomes the longest one and the blue path the shortest one. Both greedy-long and greedy-short give the same approximation ratio as choosing heaviest paths in the original flow $X_{\ell, 2^{\ell+1}}$.

By Lemma 3.2, there exists a class of instances where greedy-weight uses $\Theta(2^\ell)$ paths for the decomposition, while the optimal decomposition uses $\mathcal{O}(\ell)$ paths. Since $\Theta(m) = \Theta(2^\ell)$ for the graphs G_ℓ , we have $\Omega(\frac{2^\ell}{\ell}) = \Omega(\frac{m}{\log m})$ for the approximation ratio of greedy-weight on the class of instances $(G_\ell, X_{\ell, 2^{\ell+1}})$. $\square$

We can adjust the graphs G_ℓ to show the same lower bound for greedy-long. By subdividing every edge in the backbone path, it will become the longest in the graph. See Figure 3.3 for this graph, which we call G_ℓ^* . Greedy-long chooses the same weighted paths as greedy-weight in G_ℓ^* , and thus obtains the same approximation ratio of $\Omega(\frac{m}{\log m})$ in this class of instances. Similarly, greedy-short also obtains the same approximation ratio, as it chooses the same set of weighted paths as greedy-weight (but in opposite order).

3.2 Flow derived minors

In the following, we describe structure of DAGs, in order to understand the behaviour of the DAG width. We use a notion of minors that uses flows as a key ingredient. On a flow network (G, X) with $X \geq 0$, the process of removing weighted paths from X naturally defines the class of subgraphs of G , which can appear during the decomposition of the flow.

Definition 3.5 ([Paper I, Section 4.2]). Let $G = (V(G), E(G))$ be an s - t DAG. We call a DAG $H = (V(H), E(H))$ an f-subgraph, if there

exists a flow $X : E(G) \rightarrow \mathbb{N}$ such that $V(H) = V(G) \setminus \{v \in V(G) \mid \sum_{u \in N^-(v)} X(u, v) = \sum_{u \in N^+(v)} X(v, u) = 0\}$ and $E(H) = \text{supp}(X)$. We write $H = G|_X$.

An f-subgraph of an s - t DAG G is again an s - t DAG, and the f-subgraphs of G are the collection of subgraphs that admit an s - t flow. A flow decomposition of X on a DAG G of size k with $X \geq 0$ can be described as a sequence of flow networks $(G_0, X_0), \dots, (G_k, X_k)$ with $(G_0, X_0) = (\emptyset, 0)$ and $(G_k, X_k) = (G, X)$. Every G_{i-1} is an f-subgraph of G_i and the support of $X_i - X_{i-1}$ is the i -th weighted path in the decomposition.

Definition 3.6 (Y-to-V contractions). We consider orientated contractions of edges with respect to the edge direction:

1. An *in-contraction* of an edge (u, v) of a directed graph $G = (V, E)$ replaces G by a new graph G' , where $V' = V \setminus \{v\}$ and $E' = (E \setminus \{(x, y) \mid v \in \{x, y\}\}) \cup \{(u, y) \mid y \in V \text{ and } (v, y) \in E\}$.
2. An *out-contraction* of an edge (u, v) of a directed graph $G = (V, E)$ replaces G by a new graph G' , where $V' = V \setminus \{u\}$ and $E' = (E \setminus \{(x, y) \mid u \in \{x, y\}\}) \cup \{(x, v) \mid x \in V \text{ and } (x, u) \in E\}$.
3. We call in- and out-contractions of edges (u, v) of an s - t DAG for which $\deg^-(v) = 1$ and $\deg^+(u) = 1$, respectively, *Y-to-V contractions*.

This definition is very similar to the definition of butterfly contractions on directed graphs that was introduced by Robertson and Seymour [30] in their paper series on graph minors. Butterfly contractions are allowed under the same constraints (that is, an edge (u, v) can be butterfly contracted if $\deg^+(u) = 1$ or $\deg^-(v) = 1$). Note that unlike in butterfly contractions, which are commonly defined on simple graphs, parallel edges that occur from a butterfly contraction are merged to one edge. However, we use Y-to-V contractions on multigraphs where we keep parallel edges.

Remark 3.7. Contracting an edge (u, v) only changes the out-degree of u and the in-degree of v . This implies that the resulting graph of a sequence of Y-to-V contractions is invariant under permutations of this sequence.

Remark 3.8. Y-to-V in-contractions and Y-to-V out-contractions partition¹ the DAG into edge-disjoint out- and in-trees, respectfully. That is, a sequence of Y-to-V out-contractions of a DAG $G = (V, E)$ can be expressed

¹More precisely, they partition the edge set.

via a disjoint union $I_1 \cup \dots \cup I_k = E$ where the edges in every I_j form an in-tree. The edges in the Y-to-V contracted graph correspond to the pairs of leaf nodes and root node of every I_j . Similarly, a sequence of Y-to-V in-contractions of a DAG $G = (V, E)$ can be expressed through a partition $O_1, \dots, O_\ell$ of E , where each O_j forms an out-tree, and edges in the Y-to-V contracted graph correspond to the pairs of root node and leaf nodes of every O_j .

Y-to-V contractions are useful in the context of directed graphs for several reasons. Similarly to butterfly contractions, they do not introduce any new cycles. Moreover, they maintain flow decompositions.

Lemma 3.9 ([22], Lemma 4.1). *A flow X on an s - t DAG G can be decomposed into k weighted paths if and only if the flow X restricted to any Y-to-V contracted graph H can be decomposed into k weighted paths.*

Using f-subgraphs and Y-to-V contractions, we can define a notion of directed minors. They are compact representations of the f-subgraphs of an s - t DAG. We first use a directed minor notion that has been considered by Deligkas and Meir [8].

Definition 3.10 ([8], Definition 3). A digraph G' is a directed minor (or d-minor) of a digraph G if G' can be obtained from G by a sequence of the following operations:

1. **Deletion.** Deleting an edge $(a, b) \in E$ where $\deg^+(a) > 1$ and $\deg^-(b) > 1$.
2. **Backward contraction.** Contracting an edge $(a, b) \in E$ where $\deg^-(b) = 1$.
3. **Forward contraction.** Contracting an edge $(a, b) \in E$ where $\deg^+(a) = 1$.

This definition indeed fits our needs. The edge deletion operation aligns with the subgraphs that we can obtain during the decomposition of a flow.

Lemma 3.11 ([Paper III, Lemma 19]). *A DAG G' is a d-minor of a DAG G if and only if G' is a Y-to-V contracted graph of an f-subgraph of G .*

Important concepts of graph minor theory are the notions of a *hereditary* graph class and a *forbidden minor*, which date back to the original graph minor paper series [30]. A class of DAGs $\mathcal{G}$ is called hereditary if it is closed under d-minor operations. A hereditary class of DAGs $\mathcal{G}$ can be defined

by listing all minimal graphs² that are not contained in $\mathcal{G}$. We call them forbidden d-minors. Moreover, given two DAGs G and H , we say that G is H -d-minor free, if H is not a d-minor of G .

Definition 3.12 ([8]). The parallel-width, $\text{par-width}(G)$, of an s - t DAG G is the size of the largest minimal cut-set of G . That is, the size of a set $C \subseteq E$, such that all s - t paths cross C , and for all $C' \subsetneq C$, there exists some s - t path that avoids C' .

Deligkas and Meir [8] showed that $\text{par-width}(G) < c$ if and only if G is $G_{P(c)}$ -d-minor free, where $G_{P(c)}$ is defined as the DAG of two nodes s and t and c parallel edges $(s, t)_i$ ($i \in [c]$). Moreover, the problem of finding the $\text{par-width}(G)$ is NP-hard [8].

3.3 Width-stable graphs

In the following, we discuss the class of graphs whose width is monotone with respect to the underlying flow. As it turns out, they are a natural generalisation of series-parallel DAGs.

Definition 3.13 ([Paper II, Definition 11]). We call an s - t DAG G *width-stable*, if for all flows $0 \leq Y \leq X$, $\text{width}(G|_Y) \leq \text{width}(G|_X)$.

Many MFD instances satisfy width-stability. For example, the reduction from 3-PARTITION in Figure 2.2 creates a width-stable graph. This shows that MFD is strongly NP-hard also on width-stable instances. Moreover, practical instances are often width-stable.

Definition 3.14 ([Paper II, Section 3]). Let G be an s - t DAG and let F be a funnel that is an f-subgraph of G . A *central path* of the funnel F in G is a u - v path in G , such that u and v are in F , u is merging and v is forking. We call F a funnel with central path.

For example, central edges in Figure 3.1 are central paths of length 1. We can use central paths to recognize width-stability.

Lemma 3.15 ([Paper II, Lemma 13]). *Let G be an s - t DAG. The following statements are equivalent:*

1. G is width-stable,

²Minimal means here that the d-minors of these graphs (that are not equal to the graphs themselves) are contained in $\mathcal{G}$.

2. For every flow $X \geq 0$ on G , there exists an s - t path carrying at least $\text{Val}(X)/\text{width}(G|_X)$ flow,
3. G has no funnel f -subgraph with central path.

Proof. We will show a circular chain of three implications to show the equivalence of the statements.

(1) $\implies$ (2): We begin by showing that statement 1 implies statement 2. Assume G is an s - t DAG with flow $X \geq 0$, such that no s - t path carrying $b = \text{Val}(X)/\text{width}(G|_X)$ exists. We will show that G is not width-stable. We can assume that $G = G|_X$. Let S be the set of vertices reachable from s by a path carrying b flow. By assumption, $t \notin S$, so $(S, V \setminus S)$ is an s, t -cut and $X(u, v) < b$ for any edge (u, v) with $u \in S, v \in V \setminus S$. Define T to be the set of vertices that can reach t via a path involving vertices only from $V \setminus S$:

$$T := V \setminus \{v \in V \mid \text{all } v-t \text{ paths cross } S\}.$$

Since $S \subseteq V \setminus T$, $(V \setminus T, T)$ is an s, t -cut. Note that if (u, v) is an edge with $u \in V \setminus T$ and $v \in T$, then also $u \in S$ and $v \in V \setminus S$ and thus $X(u, v) < b$. The set C defined by

$$C := \{(u, v) \in E \mid u \in V \setminus T, v \in T\} = E \cap (S \times T)$$

is an s, t cut-set (a set of edges that every s - t path has to cross) and we have $X(e) < b$ for every $e \in C$. This implies $|C| > \text{width}(G)$.

We construct a new flow $Y \geq 0$ on G for which C is an antichain in $G|_Y$. Define the following paths in G :

- $p_s(u)$ for all $u \in S$: an s - u path with all nodes in S ,
- $p_t(v)$ for all $v \in T$: a v - t path with all nodes in T .

Consider the path cover

$$\mathcal{P} = \{p_s(u) \cup \{(u, v)\} \cup p_t(v) \mid (u, v) \in C\}$$

and let Y be the induced flow³ of $\mathcal{P}$. Note that $Y(e) = 0$ for all $e \in T \times S$. The set C is an antichain in $G|_Y$, since passing two edges of C requires using an edge in $T \times S$. Finally, $X + Y \geq Y$, but $\text{width}(G|_{X+Y}) = \text{width}(G) < |C| \leq \text{width}(G|_Y)$.

³Recall that the induced flow Y of a path cover $(P)_1^\ell$ is defined as $Y = P_1 + \dots + P_\ell$, where we identify each path P_i with a 0/1-flow with value 1 on the path edges (i.e., the characteristic function of P_i).

(2) $\implies$ (3): Now we prove that statement 2 implies statement 3. Assume that G has a funnel subgraph F with a central path P . Let C be a maximum antichain of F . We define flows Z_{stable} , Z_{unstable} and Z :

- Z_{stable} is defined to be the flow induced by the minimum path cover of F^4 .
- Z_{unstable} is defined as the flow induced by the following path cover of the graph consisting of F and P : One s - t path goes along P , covering two edges in C^5 , and the other paths cover F avoiding P , this is possible with additional $|C| - 2$ paths.
- Finally, $Z := Z_{\text{stable}} + Z_{\text{unstable}}$.

We have $\text{Val}(Z) = \text{Val}(Z_{\text{stable}}) + \text{Val}(Z_{\text{unstable}}) = |C| + (|C| - 1) = 2|C| - 1$ and $\text{width}(G|_Z) \leq |C| - 1$, and thus

$$\frac{\text{Val}(Z)}{\text{width}(G|_Z)} \geq \frac{2|C| - 1}{|C| - 1} > 2,$$

but all s - t paths in $G|_Z$ carry no more flow than 2, because C is an s , t -cut set of $G|_Z$ and all Z flow weights on C are 2.

(3) $\implies$ (1): Finally, we show that statement 3 implies statement 1. For that, we need to define the concept of *splicing* paths. This concept has been used before (see, for example, [6]).

Definition 3.16. Let P, Q be paths that both traverse the same node v , that is,

$$P = \{(u_{P,1}, u_{P,2}), (u_{P,2}, u_{P,3}), \dots, (u_{P,i-1}, v), (v, u_{P,i+1}), \dots\}$$

and

$$Q = \{(u_{Q,1}, u_{Q,2}), (u_{Q,2}, u_{Q,3}), \dots, (u_{Q,j-1}, v), (v, u_{Q,j+1}), \dots\}.$$

Then the operation $\text{splice}(P, Q, v)$ produces the two paths

$$P' = \{(u_{P,1}, u_{P,2}), (u_{P,2}, u_{P,3}), \dots, (u_{P,i-1}, v), (v, u_{Q,j+1}), \dots\}$$

and

$$Q' = \{(u_{Q,1}, u_{Q,2}), (u_{Q,2}, u_{Q,3}), \dots, (u_{Q,j-1}, v), (v, u_{P,i+1}), \dots\}.$$

⁴Funnel admits a unique flow decomposition, and in particular also a unique minimum s - t path cover.

⁵Indeed, this path covers *exactly* two edges in C : one edge needs to be covered to reach P and another edge must be covered to reach t , and since G is a DAG, we cannot use the same edge twice.

Assume that $G = (V, E)$ is not width-stable, let $X \geq Y \geq 0$ be flows on G with $\text{width}(G|_Y) > \text{width}(G|_X)$ and let $\bar{C}$ be a maximum antichain of $G|_Y$. Let F be a funnel f-subgraph of $G|_Y$ containing $\bar{C}$ as maximum antichain, and let C be the rightmost maximum antichain of F (that is, the head of every edge in C is t or is merging). Consider a minimum path cover $\mathcal{P}$ of $G|_X$.

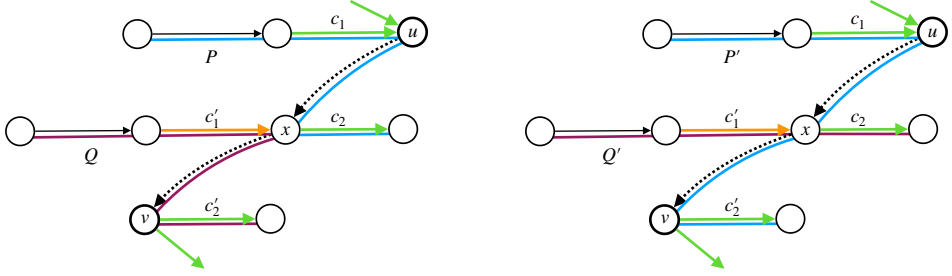
Using this path cover, we construct a path in $G|_X$ from a merging node in F to a forking node in F . Since C is not an antichain of $G|_X$, there must be a path P in $\mathcal{P}$ traversing the edges $c_1, \dots, c_\ell$ in C , where $\ell \geq 2$, and it traverses a merging node (due to the assumption of C being rightmost, the head of c_1 is merging). If P does not enter a forking node of F , then there exists a different maximum antichain of F , which gets traversed by P only once: intuitively, to find it, we move every edge c_i for $i = 2, \dots, \ell$ to the left. Formally, each tail u_i of c_i , for $i = 2, \dots, \ell$, has $\deg^-(u_i) = 1$, so we can replace c_i by its unique predecessor edge. We can do that until c_i is no longer covered by P , because P does not enter a forking node. This gives us a new maximum antichain C' of F .

The path P only covers one edge of C' , while the invariant that some path from $\mathcal{P}$ must cross edges $c'_1, \dots, c'_{\ell'}$ of C' , where $\ell' \geq 2$, remains true, since the graphs $G|_X$ and F and thus their widths did not change. Let Q now be such a path, and note that it traverses a merging node or, if it does not, crosses the tail of P , in the case where c'_1 is the unique predecessor of a previous c_j for some $j = 2, \dots, \ell$. If we splice P and Q on the tail of c_j (equivalently, the head of c'_1), then we gain two new paths P' and Q' , where P' is a path that traverses a merging node and covers $c_1, \dots, c_{j-1}, c'_2, \dots, c'_{\ell'}$, where $c_1, \dots, c_{j-1} \in C$ and $c'_2, \dots, c'_{\ell'} \in C'$ (see Figure 3.4). This set of edges also forms a maximum antichain of F . We can repeat this process finitely many times until we obtain a path that traverses a forking node (and a merging node, which holds for all intermediate paths). $\square$

Lemma 3.15 provides a graph structural classification for width-stable graphs. We will use it to show a strong greedy-weight approximation upper bound on width-stable graphs. The following DAG will serve as a forbidden d-minor.

Definition 3.17 ([Paper III, Definition 20]). The graph Ch_k is defined to be the DAG consisting of 4 vertices s, u, v and t , and the following edges: k parallel edges (s, u) , k parallel edges (v, t) , 1 edge (u, v) , 1 edge (u, t) and 1 edge (s, v) .

See Figure 3.5 for an example. The Braess graph [4], which serves as a common example to show certain properties related to transportation planning, is equal to Ch_1 . It is also the forbidden d-minor for series-parallel



(a) Two of the green edges, c_1 and c_2 , are traversed by the blue path P . Since the blue path does not traverse a forking node, we consider the orange edge c'_1 which is the unique predecessor of c_2 . The purple path Q traverses c'_1 and c'_2 .

(b) The blue and purple paths P and Q from Figure 3.4a are spliced at the node x , resulting in new paths P' and Q' . The blue path P' traverses both a merging node (u) and a forking node (v).

Figure 3.4: Visualisation of the proof of Lemma 3.15, 3. $\implies$ 1. The dotted edges indicate a central path. The green edges each form an antichain in the funnel. If the edge c_2 is replaced by c'_1 , the edges remain an antichain in the funnel.

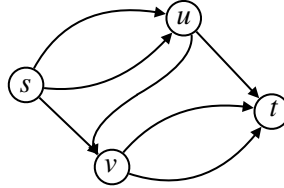


Figure 3.5: The graph Ch_2 .

DAGs. Since $\text{Ch}_k \subseteq \text{Ch}_{k+1}$, every Ch_k -d-minor free graph is also Ch_{k+1} -d-minor free.

Lemma 3.18 ([Paper III, Lemma 21]). *An s - t DAG G is width-stable if and only if it is Ch_2 -d-minor free.*

Proof sketch. Assume G contains Ch_2 as d-minor. Ch_2 itself is not width-stable by Lemma 3.15, and thus G is not width-stable.

Assume G is not width-stable. By Lemma 3.15, it contains a funnel f-subgraph with central path. This subgraph contains Ch_2 as d-minor: let the central path start at the merging node u and end in the forking node v . Since u is merging, we can find at least two paths from s to u , and similarly we can find at least two paths from v to t . The f-subgraph with the induced flow of this path cover V-to-Y contracts to Ch_2 . $\square$

Corollary 3.19 ([Paper III, Corollary 22]). *It can be verified in polynomial time if an s - t DAG G is width-stable.*

Proof. From Lemma 3.18 it follows that to see if G is width-stable, we need to find a Ch_2 d-minor of G or deduce that it is not contained in G . Deligkas and Meir [8] showed that this can generally be done in polynomial time, parameterised by the size of the minor. In this case, there exists also a simple algorithm that does not use previous results.

Consider every pair of nodes u, v , such that there exists a u - v path in G . We must verify that an f-subgraph of G can Y-to-V contract to Ch_2 with respect to the nodes s, u, v and t . This can be achieved by simple reachability questions.

Consider the subgraph of G that consists of all nodes and vertices that are traversed by some s - u path. By the definition of Ch_2 , there must be two separate paths from s to u . The graph intersecting with these two paths then contains $G_{P(2)}$ as d-minor. We can analogously check the reachability between v and t . If we delete u from G , we can check whether there exists a s - v path that does not traverse u . Doing so analogously for u and t , we have shown that Ch_2 is a d-minor of G , with the 2 chosen internal nodes u and v . $\square$

Width-stability is a graph structural feature that impacts the approximation of flow decomposition. We start by demonstrating an improved approximation factor upper bound for greedy-weight. The following theorem is the first theoretical explanation for the good experimental performance of practical implementations of saturating MFD solvers [2, 29, 35].

Theorem 3.20 ([Paper II, Theorem 2]). *Greedy-weight is a $O(\log \text{Val}(X))$ -approximation algorithm on width-stable DAGs.*

Proof. Let G be width-stable. According to Lemma 3.15, for every flow $X \geq 0$, there exists an s - t path carrying $\text{Val}(X)/\text{width}(G|_X)$ flow. Let $b := \text{width}(G|_X)$. Greedy-weight thus always chooses a path of at least this weight. After x iterations, the flow $X^{(x)}$ that is left has value at most $\text{Val}(X^{(x)}) = \text{Val}(X) \left(\frac{b-1}{b}\right)^x$, since $\text{width}(G|_Y) \leq b$ for all $Y \leq X$. If $\text{Val}(X^{(x)}) < 1$, then $X^{(x)} = 0$, since all flow weights are integer. Solving for x , we obtain

$$x > \frac{\log(\text{Val}(X))}{\log\left(\frac{b}{b-1}\right)}.$$

The approximation ratio is thus

$$\frac{x}{\text{mfd}_{\mathbb{N}}(G, X)} \leq \frac{O(\log(\text{Val}(X)))}{b \log\left(\frac{b}{b-1}\right)} = O(\log(\text{Val}(X))),$$

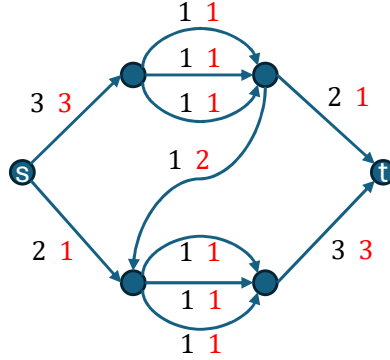


Figure 3.6: [Paper III, Figure 1] An example of minimally covering flow (left) and minimum flow (right). Note that the value of the left flow (5) is larger than the value of the right flow (4), despite being smaller in the central edge.

since $b = \text{width}(G|_X) \leq \text{mfd}_{\mathbb{N}}(G, X)$ for all $X \geq 0$, and $b \log \left(\frac{b}{b-1} \right) = O(1)$ for all $b \geq 2$. $\square$

3.4 Flow-width

In Chapter 4, we want to use a small number of paths to decompose a large portion of the flow. The value $\text{width}(G)$ – as a lower bound to $\text{mfd}_{\mathbb{N}}(G, X)$ for all $X > 0$ – is a good candidate, as we show in Section 4.1, but contains a small caveat in $\text{MFD}_{\mathbb{N}}$. There might exist an edge $e \in E$, such that every minimum path cover of G covers e at least μ times, while it is possible to define flows $X > 0$ with $X(e) < \mu$ (see Figure 3.6 for an example). We mitigate this issue by defining the width of a flow network (G, X) .

Definition 3.21 ([Paper III, Definition 1] Flow-width). Let G be a DAG and $X \geq 0$ be a flow on G . We define the *flow-width* of G and X , $\text{fw}(G, X)$, as the smallest number of paths satisfying the following properties:

1. **Covering:** Every edge $e \in E(G)$ with $X(e) > 0$ appears in at least one path, and
2. **Upper bounds:** Every edge $e \in E(G)$ appears in at most $X(e)$ paths.

Similarly to the case of $\text{width}(G)$, we can find $\text{fw}(G, X)$, and a path cover of this size, by subtracting augmenting paths from X . Such path covers are

not necessarily minimum path covers of G , but they are always *minimal*. While the DAG width corresponds to the value of a minimum flow, where the upper bounds on the edges are ∞ , the flow-width corresponds to the value of minimally covering flows, where the upper bounds on the edges are defined by a flow.

Definition 3.22 ([Paper III, Section 3.1]). We call a flow *minimally covering*, if every s - t path carries at most weight 1, and we call a path cover *minimally covering* if its induced flow is minimally covering.

Lemma 3.23 ([Paper III, Lemma 13] Properties of flow-widths). *Let G be an s - t DAG.*

1. *For all flows $X \geq 0$, $\text{width}(G|_X) \leq \text{fw}(G, X) \leq \text{mfd}_{\mathbb{N}}(G, X)$,*
2. *For all flows $Y \geq X \geq 0$ with $G|_Y = G|_X$, $\text{fw}(G, X) \geq \text{fw}(G, Y)$.*

Recall that a DAG G is width-stable if $\text{width}(G|_X) \leq \text{width}(G|_Y)$ for all flows $0 \leq X \leq Y$. We can give an equivalent definition using the flow-width.

Lemma 3.24 ([Paper III, Lemma 16]). *Let G be an s - t DAG.*

1. *If G is width-stable, then $\text{fw}(G, X) = \text{width}(G|_X)$ for all flows $X \geq 0$ on G .*
2. *G is width-stable if and only if $\text{fw}(G, X) \leq \text{fw}(G, Y)$ for all flows $0 \leq X \leq Y$.*

The following lemma shows that the flow-width combines two width measures of the underlying graph: the $\text{width}(G)$ (i.e., the minimum number of paths to cover all the edges) and the $\text{par-width}(G)$ (i.e., the largest minimal cut-set). It also shows that minimally covering flows can be decomposed using at most $\text{par-width}(G)$ weighted paths.

Lemma 3.25 ([Paper III, Lemma 4]). *For all s - t DAGs G , we have*

$$\text{width}(G) = \min\{\text{fw}(G, X) \mid X > 0\}$$

and

$$\text{par-width}(G) = \max\{\text{fw}(G, X) \mid X \geq 0\}.$$

Chapter 4

Approximation and hardness of Minimum Flow Decomposition

In this chapter, we provide an approximation algorithm for the Minimum Flow Decomposition problem for both the non-negative and general integer variants. The approach differs from the previous mentioned framework of saturating algorithms, including, for example, greedy-weight. Given a flow X on a DAG G , our aim is to express X as the sum of weighted flows $X = w_1 Y_1 + \dots + w_\ell Y_\ell$, where ℓ is a small integer and each Y_i for $i \in [\ell]$ can be easily decomposed using a small number of paths. We split X into $\ell \leq \lceil \log \|X\| \rceil + 1$ flows with weights in $\{2^i \mid 0 \leq i \leq \lceil \log \|X\| \rceil\}$ using a parity fixing approach.

We demonstrate the role of two graph parameters, the width of G (i.e., the minimum number of paths to cover all edges) and the parallel-width of G (i.e., the largest minimal cut-set). For a generalised problem, $\text{MCCD}_{\mathbb{Z}}$, we show that this approach gives an approximation factor of $\lceil \log \|X\| \rceil + 1$. On the other hand, adapting it to $\text{MFD}_{\mathbb{N}}$ comes with issues that need to be addressed. We obtain an approximation algorithm for a large class of instances that are particularly relevant in practical applications.

4.1 Width helps approximating $\text{MCCD}_{\mathbb{Z}}$

In this section, we consider a generalised version of $\text{MFD}_{\mathbb{Z}}$ on graphs with cycles, and without a source or a sink node. In addition to the graph $G = (V, E)$, we are given a cost function $c: E \rightarrow \mathbb{R}_{\geq 0}$ and we use the notation $G = (V, E, c)$. The *cost* of a circulation X is defined as $\tilde{c}(X) = \sum_{e \in E} c(e)X(e)$. Note that $\tilde{c}(\cdot)$ is a linear function: $\tilde{c}(X + Y) = \tilde{c}(X) + \tilde{c}(Y)$ for any two circulations X, Y on G .

Definition 4.1 ([Paper II, Definition 17]). Given a tuple (G, X) of a graph with cost function $G = (V, E, c)$ and a circulation $X: E \rightarrow \mathbb{Y}$, a *circulation decomposition* of cost k of X is a family of circulations $Y_i: E \rightarrow \mathbb{N}$ with weights $(w_1, \dots, w_\ell) \in \mathbb{Y}^\ell$ such that $X = w_1 Y_1 + \dots + w_\ell Y_\ell$ and $\tilde{c}(Y_1 + \dots + Y_\ell) = k$.

Instead of expressing a flow into flows of small decomposition size, we decompose the circulation into circulations of small cost. We denote by $\text{mccd}_{\mathbb{Y}}(G, X)$ the smallest cost of a circulation decomposition of X with weights in $\mathbb{Y}$, and we call the problem of finding a circulation decomposition of cost $\text{mccd}_{\mathbb{Y}}(G, X)$ the *Minimum Cost Circulation Decomposition* ($\text{MCCD}_{\mathbb{Y}}$) problem. $\text{MCCD}_{\mathbb{Y}}$ generalises MFD, as we can add an edge (t, s) to any s - t DAG and attach to it cost 1. A consequence is the following lemma, we refer to Paper II for a formal proof.

Lemma 4.2 ([Paper II, Lemma 18]). *$\text{MCCD}_{\mathbb{Y}}$ is NP-hard.*

We must also generalise the width of a DAG to graphs with cycles and costs. Instead of minimising the cardinality, we again minimise the cost. Such a minimum cost circulation cover can, similarly to a minimum path cover of DAGs, also be found in polynomial time.

Definition 4.3 ([13], [Paper II, Definition 19]). Given $S \subseteq E$ and a graph G , we call a circulation C of minimum cost satisfying $C(e) \geq 1$ for all $e \in S$ minimum cost circulation cover of S , and we write $\text{mccc}_S(G) = \tilde{c}(C)$. If $S = E$, we use $\text{mccc}(G)$ instead.

Lemma 4.4 ([13], Theorem 3.6). *Let $G = (V, E, c)$ be a graph, and $S \subseteq E$. A minimum cost circulation cover of S can be computed in $O(n \log m(m + n \log n))$ time.*

To achieve an approximation factor for $\text{MCCD}_{\mathbb{Z}}$ of $\lceil \log \|X\| \rceil + 1$, we show that every circulation X can be expressed as a sum $X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i Y_i$, such that every Y_i has a cost of at most $\text{mccc}(G)$. We note that $\text{mccc}(G)$ is not necessarily a lower bound to $\text{mccd}_{\mathbb{Z}}(G, X)$ in the case where $\text{supp}(X) \neq E$. We will show how to handle this issue later.

We use two key routines in our algorithm. Given a circulation X , we first find a *unitary* circulation Y (a circulation with weights in $\{-1, 0, 1\}$), such that $X + Y \equiv_2 0$. We call this routine the *parity fixing* of X . Indeed, such a circulation Y always exists.

Lemma 4.5 ([Paper II, Lemma 21]). *For any circulation $X: E \rightarrow \mathbb{Z}$ on $G = (V, E, c)$, there exists a circulation $Y: E \rightarrow \mathbb{Z}$ such that $X \equiv_2 Y$ and $\|Y\| \leq 1$.*

Algorithm 1 [Paper II, Algorithm 1] $\text{UNITARY}(G, X)$: Produces a unitary circulation Y given an input circulation X such that $X \equiv_2 Y$, as in Lemma 4.5.

```

1:  $E_{\text{odd}} \leftarrow$  odd edges of  $G$ , undirected
2:  $C \leftarrow$  a partition of  $G_{\text{odd}} = (V, E_{\text{odd}})$  into cycles, oriented arbitrarily
3:  $E_{\text{odd}}^+ \leftarrow$  directed edges of  $C$ 
4: for  $(u, v) \in E$  do
5:   if  $(u, v) \in E_{\text{odd}}^+$  then
6:      $Y(u, v) \leftarrow +1$ 
7:   else if  $(v, u) \in E_{\text{odd}}^+$  then
8:      $Y(u, v) \leftarrow -1$ 
9:   else
10:     $Y(u, v) \leftarrow 0$ 
11:   end if
12: end for
13: return  $\mu$ 

```

We can find the unitary circulation Y by observing that the set of the odd parity edges consists of s - t cycles, whose direction is omitted, due to the conservation of flow. We refer to Algorithm 1 for the procedure. We can replace the parity fixing flow Y by $-Y$, which flips the sign of the cost of Y and maintains the parity fixing property of the flow.

Corollary 4.6 ([Paper II, Corollary 22]). *Any circulation $X: E \rightarrow \mathbb{Z}$ can be written as $X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i$, where $Y_i: E \rightarrow \mathbb{Z}$ is a circulation with $\|Y_i\| \leq 1$ for all i .*

The second step is to decompose each unitary circulation Y obtained from Lemma 4.5. We use a cost of at most $\text{mccc}(G)$.

Lemma 4.7 (Adaptation from [Paper II, Lemma 23]). *For a circulation $X: E \rightarrow \mathbb{Z}$, $\|X\| \leq 1$ on a graph $G = (V, E, c)$, we define two circulations $X^+, X^-: E \rightarrow \mathbb{Z}$ by*

$$\begin{aligned}
X^+ &:= (C - D + X)/2, \\
X^- &:= (C - D - X)/2,
\end{aligned}$$

where C is a circulation with $C \geq 1$ and $\tilde{c}(C) = \text{mccc}(G)$, and is chosen according to Lemma 4.4. The circulation $D = \text{UNITARY}(G, X + C)$ is chosen according to Lemma 4.5 with $\tilde{c}(D) \geq 0$. We have

1. $X^+, X^- \geq 0$,

2. $X = X^+ - X^-$,
3. $\tilde{c}(X^+) + \tilde{c}(X^-) \leq \text{mccc}(G)$.

By combining Corollary 4.6 and Lemma 4.7 we obtain the following result. We can use it to define the parity fixing approximation algorithm for $\text{MCCD}_{\mathbb{Z}}$.

Theorem 4.8 ([Paper II, Theorem 24]). *Given a graph $G = (V, E, c)$ and a circulation $X: E \rightarrow \mathbb{Z}$ with $k := \lceil \log \|X\| \rceil$, there exist circulations Y_i^+, Y_i^- for $i = 0, \dots, k$ and weights $\{w_0, \dots, w_k\} \subseteq \{2^i \mid i \in \mathbb{N}\}$, with $\tilde{c}(Y_0^+ + \dots + Y_k^+ + Y_0^- + \dots + Y_k^-) \leq (k + 1) \cdot \text{mccc}(G)$ such that $X = w_0(Y_0^+ - Y_0^-) + \dots + w_k(Y_k^+ - Y_k^-)$.*

As mentioned above, $\text{mccc}(G)$ is not necessarily a lower bound to $\text{mccd}_{\mathbb{Z}}(G, X)$. Clearly, this is the case when $X = 0$ and G is a non-empty graph, for example. We mitigate this issue by replacing G with a subgraph of G . Let $\mathcal{S} = \text{supp}(X)$ and find a minimum circulation cost cover $\mathcal{C}$ in G covering $\mathcal{S}$, according to Lemma 4.4. Create the subgraph G' by removing all edges that are not covered by $\mathcal{C}$. We then have $\text{mccc}(G') \leq \text{mccd}_{\mathbb{Z}}(G, X)$, and we can simply replace G by G' . The complete procedure is given in Algorithm 2, which contains a call of Algorithm 1.

Theorem 4.9 ([Paper II, Theorem 3]). *Algorithm 2 is a $\log \lceil \|X\| \rceil + 1$ -factor approximation algorithm in runtime $O(n \log m(m + n \log n) + m \log \|X\|)$.*

We obtain an equivalent result for the special case of decomposing integer flows on s - t DAGs. We can reduce the problem to $\text{MCCD}_{\mathbb{Z}}$ with the construction used in Lemma 4.2. The runtime improves slightly, as we can use Lemma 2.4 instead of Lemma 4.4. Note that we obtain the factor $\text{width}(G)$, since we now decompose the flows obtained by Lemma 4.7 into at most $\text{width}(G)$ paths.

Theorem 4.10 ([Paper I, Theorem 11]). *Algorithm 2 is a $\log \lceil \|X\| \rceil + 1$ -approximation for $\text{MFD}_{\mathbb{N}}$ with runtime $O(m(n + \text{width}(G) \cdot \log \|X\|)) \leq O(m^2 \log \|X\|)$.*

4.2 Parity fixing approximation for $\text{MFD}_{\mathbb{N}}$

We now proceed to analyse the parity fixing approximation algorithm for $\text{MFD}_{\mathbb{N}}$ on s - t DAGs G . Mumey et al. [25] originally implemented this algorithm for $\text{MFD}_{\mathbb{N}}$, but did not analyse the size of the decomposition of

Algorithm 2 [Paper II, Algorithm 2] $\text{PARITY-APPROX}_{\mathbb{Z}}(G, X)$: Finds a parity fixing flow decomposition as in Theorem 4.8.

```

1: Compute a minimum cost circulation cover of  $\{(u, v) \in E \mid X(u, v) \neq 0\}$  {Lemma 4.4}
2: Remove from  $G$  any edge not covered by this circulation to obtain  $G'$ 
3:  $\mathcal{P} \leftarrow \square$ ,  $w \leftarrow \square$  {length-zero vectors}
4:  $C \leftarrow$  circulation of cost  $\text{mccc}(G')$ ,  $C \geq 1$  {Lemma 4.4}
5:  $D \leftarrow \text{UNITARY}(G', C)$ ; if  $\tilde{c}(D) < 0$  set  $D = -D$  {Algorithm 1}
6:  $i \leftarrow 0$ 
7: while  $\|X\| > 1$  do
8:    $Y_i \leftarrow \text{UNITARY}(G', X)$  {Algorithm 1}
9:    $X \leftarrow (X - Y_i)/2$ 
10:   $i \leftarrow i + 1$ 
11: end while
12:  $Y_i \leftarrow X$ 
13: for  $j \in \{0, \dots, i\}$  s.t.  $Y_j \neq 0$  do
14:    $A \leftarrow C - D + Y_j$ ,  $B \leftarrow C - D - Y_j$ 
15:   Concatenate  $A$  and  $B$  to  $\mathcal{P}$ 
16:   Concatenate  $2^j$  and  $-2^j$  to  $w$ 
17: end for
18: return  $(\mathcal{P}, w)$ 

```

each Y_i . Instead, in addition to experimentally verifying that the algorithm performs well on practical inputs, Mumey et al. [25] directly showed a rather large approximation ratio upper bound of $O(\lambda^{\log \|X\|} \cdot \log \|X\|)$, where λ is the longest path in G . We will present their algorithm and provide the graph structural analysis that shows on which classes of instances the algorithm performs well and on which class of instances it might not, which marks the instances where an approximation algorithm of good ratio is still open. Given a flow $X \geq 0$, we aim find a flow $Y \geq 0$, such that $X \geq Y$ and $X - Y \equiv_2 0$. Mumey et al. [25] showed that such a flow can be found by solving the following Integer Linear Program (ILP), which models a min-flow problem:

$$\begin{aligned}
& \text{minimise } \text{Val}(Y), \text{ subject to} \\
& Y \text{ is an integer flow on } G, \\
& 0 \leq Y \leq X, \\
& Y(e) \geq 1 \text{ for all } e \in E(G) \text{ where } f(e) \text{ is odd.}
\end{aligned} \tag{4.1}$$

Algorithm 3 $\text{PARITY-APPROX}_{\mathbb{N}}(G, X)$: Finds a parity fixing flow decomposition for $\text{MFD}_{\mathbb{N}}$ [25].

```

1:  $\mathcal{P} \leftarrow \emptyset, w \leftarrow \emptyset$  {length-zero vectors}
2:  $i \leftarrow 0$ 
3: while  $\text{Val}(X) > 0$  do
4:    $Y \leftarrow$  Optimal solution of Problem 4.1 for  $(G, X)$ 
5:   Naively decompose  $Y$  into  $\text{Val}(Y)$  paths; concatenate to  $\mathcal{P}$ 
6:   Concatenate  $\text{Val}(Y)$  copies of  $2^i$  to  $w$ 
7:    $X \leftarrow (X - Y)/2$ 
8:    $i \leftarrow i + 1$ 
9: end while
10: return  $(\mathcal{P}, w)$ 
```

Lemma 4.11 ([25], Lemma 1). *Every optimal solution Y of the ILP 4.1 on a given flow $X \geq 0$ fulfils $X \equiv_2 Y$.*

Clearly, solutions of Problem 4.1 are minimally covering flows. This implies that we can bound their flow values by the flow-width.

Lemma 4.12 ([Paper III, Lemma 17]). *We have $\text{Val}(Y) \leq \text{fw}(G, X)$ for every optimal solution Y of the ILP 4.1 on a given flow $X \geq 0$.*

We obtain a result that is equivalent to Corollary 4.6 in the following way. We express the input flow X as the sum of weighted flows that we aim to decompose into a small number of paths. We do this by fixing the parity of the flow, using at most $\lfloor \log \|X\| \rfloor + 1$ weighted flows in the sum, and we can decompose each of them into at most a flow-width number of paths by Lemma 4.12.

Theorem 4.13 ([Paper III, Theorem 2]). *Given a flow $X \geq 0$ on a DAG G , we can decompose the flow into $\lfloor \log \|X\| \rfloor + 1$ flows Y_i , such that*

$$X = \sum_{i=0}^{\lfloor \log \|X\| \rfloor} 2^i Y_i \quad \text{and} \quad \text{Val}(Y_i) \leq \text{fw}(G, Y^{(i)}), \quad (4.2)$$

where $Y^{(0)} = X$ and $Y^{(i)} = X - \sum_{j=0}^{i-1} 2^j Y_j$. This can be done in time $O(nm \log \|X\|)$.

We complete the approximation algorithm by decomposing each Y_i into at most $\text{fw}(G, Y^{(i)})$ weighted paths, see Algorithm 3 for a detailed description. To obtain a good approximation ratio from this flow expression, we

must thus find upper bounds for the values $\text{fw}(G, Y^{(i)})$. On width-stable DAGs, we have $\text{fw}(G, Y) \leq \text{width}(G|_X)$ for all $0 \leq Y \leq X$ by Lemma 3.24. As a first result, we obtain a good ratio for width-stable DAGs.

Theorem 4.14 ([Paper III, Theorem 3]). *On width-stable DAGs we can approximate $\text{MFD}_{\mathbb{N}}$ given the input (G, X) with ratio $\lfloor \log \|X\| \rfloor + 1$ in runtime $O(nm \log \|X\|)$.*

We can generalise Theorem 4.14 using the parallel-width of G . Theorem 4.13 and Lemma 3.25 imply that we can decompose any flow with at most $(\lfloor \log \|X\| \rfloor + 1) \cdot \text{par-width}(G)$ weighted paths. Moreover, since $\text{width}(G|_X) \leq \text{mfd}_{\mathbb{N}}(G, X)$, we obtain the following approximation ratio.

Theorem 4.15 ([Paper III, Theorem 5]). *Given a flow network (G, X) with $X > 0$, we can approximate $\text{MFD}_{\mathbb{N}}$ with a ratio of $\left(\frac{\text{par-width}(G)}{\text{width}(G)} \cdot (\lfloor \log \|X\| \rfloor + 1) \right)$ in runtime $O(nm \log \|X\|)$. Moreover, for flow networks with $\text{mfd}_{\mathbb{N}}(G, X) \geq \text{par-width}(G)$, the ratio is $\lfloor \log \|X\| \rfloor + 1$.*

Furthermore, we can use the obtained upper bound $k := \text{mfd}_{\mathbb{N}}(G, X) \leq (\lfloor \log \|X\| \rfloor + 1) \cdot \text{par-width}(G)$ to analyse $\text{MFD}_{\mathbb{N}}$'s parameterised computational complexity. For a constant $c \in \mathbb{N}$, consider the class of $G_{P(c+1)}$ -d-minor free DAGs, that is, the class of DAGs with $\text{par-width}(G) \leq c$. Recall that $\text{MFD}_{\mathbb{N}}$ with the optimal solution k as parameter is in FPT with runtime $2^{O(k^2)} \cdot (n + \log \|X\|)$ [22]. Applying the upper bound to this runtime, we obtain $\|X\|^{O(\log \|X\|)} \cdot (m + \log \|X\|)$, which shows the following result.

Theorem 4.16 ([Paper III, Corollary 6]). *Let $c \in \mathbb{N}$ be a constant. $\text{MFD}_{\mathbb{N}}$ on (G, X) can be solved in quasi-polynomial time $\|X\|^{O(\log \|X\|)} \cdot (m + \log \|X\|)$ when $\text{par-width}(G) \leq c$ and X is coded in unary.*

In the above corollary, we assume that X is not coded in binary (in space $O(m \cdot \log \|X\|)$), but is coded with unary bits in space $O(m \cdot \|X\|)$. This is relevant in practical applications where the flow encodes, for example, information routed through a network or the captured reads of a biological sequence.

We now show that the parity fixing algorithm admits, in the worst case, an approximation factor of $\Omega(\|X\|)$. We do this by showing that the values $\text{fw}(G, X^{(i)})$ can grow very large. Consider the class of instances $(G_{(k,\ell)}, X_k)$ defined in Figure 4.1.

Lemma 4.17 ([Paper III, Lemma 23]). *For all $c > 1$, there exist $k, \ell > 0$, such that*

$$\frac{\text{par-width}(G_{(k,\ell)})}{\text{mfd}_{\mathbb{N}}(G_{(k,\ell)}, X_k)} > c.$$

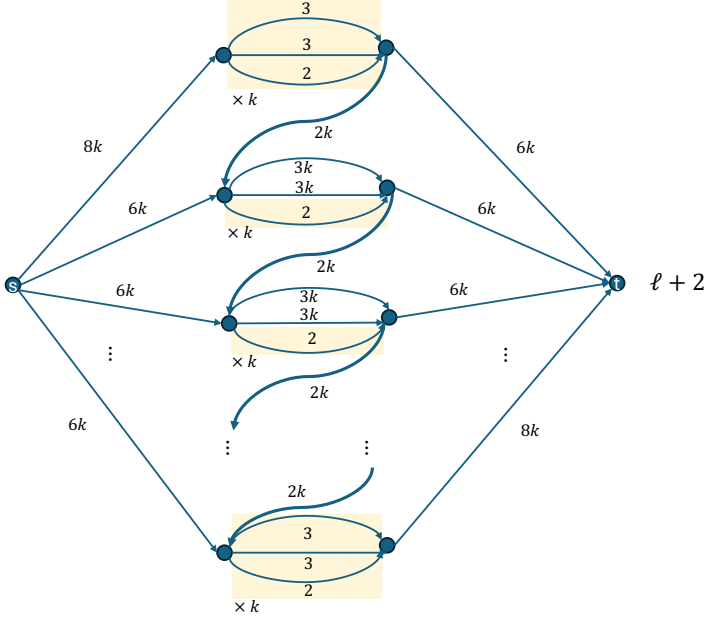


Figure 4.1: [Paper III, Figure 3] MFD instance $(G_{(k,\ell)}, X_k)$. The edges highlighted in yellow are copied k times, and there are $\ell + 2$ total central gadgets (that is, $\deg^+(s) = \ell + 2$). Thus, $\text{par-width}(G_{(k,\ell)}) = 6k + \ell(k + 2)$, and we call these edges the maximum cut-set edges. There are two central gadgets consisting of $3k$ maximum cut-set edges and ℓ central gadgets each consisting of $k + 2$ maximum cut-set edges.

Lemma 4.18 ([Paper III, Lemma 24]). *In the worst case, Algorithm 3 is a factor $\Omega(\|X\|)$ approximation algorithm.*

Proof sketch. Note that $\text{mfd}_{\mathbb{N}}(G_{(k,\ell)}, X_k) \leq 5k + 2\ell$. Let $\ell = k$. If in the first iteration, all paths avoid the edges of flow weight 2, the approximation algorithm uses $6k + k^2$ paths in the second iteration. The result follows with $k = \Theta(\|X_k\|)$. $\square$

4.3 Hardness of MFD

We now want to show lower bounds in the runtime of solving $\text{MFD}_{\mathbb{N}}$ exactly. First, we consider the computational complexity of $\text{MFD}_{\mathbb{N}}$ using width parameters. Afterwards, we disprove a conjecture stated by Kloster et al. [22]. If the conjecture had turned out to be true, it would have implied that a restricted version of $\text{MFD}_{\mathbb{N}}$, where paths are already given and we are tasked

with finding corresponding weights, can be solved in polynomial time if the paths come from a minimum flow decomposition.

4.3.1 Computational complexity of $\text{MFD}_{\mathbb{N}}$

Previous NP-hardness proofs for $\text{MFD}_{\mathbb{N}}$ use reductions to instances (G, X) that are series-parallel and have arbitrary width. Additionally, $\text{MFD}_{\mathbb{N}}$ is in FPT with the optimal solution as parameter, which implies the improved computational complexity in Theorem 4.16 for constant parallel-width graphs with unary coded flows. Moreover, instances (G, X) where $\text{width}(G) = \text{par-width}(G) = 1$ are just weighted paths, and admit $\text{mfd}_{\mathbb{N}}(G, X) = 1$. This motivates the question whether $\text{MFD}_{\mathbb{N}}$ remains NP-hard with constant (parallel-)width. We answer this question in a tight manner:

1. $\text{MFD}_{\mathbb{N}}$ is *strongly* NP-hard on DAGs G of width 3.
2. $\text{MFD}_{\mathbb{N}}$ is NP-hard on DAGs G of width 2.

Note that $\text{width}(G) = 2 \iff \text{par-width}(G) = 2$, since every DAG of width or parallel-width 2 is Ch_2 -d-minor free. By Lemma 3.18, such DAGs are width-stable, which implies that the parallel-width and the width are equal.

Theorem 4.19 ([Paper III, Theorem 7]). *$\text{MFD}_{\mathbb{N}}$ on (G, X) is strongly NP-hard even when $\text{width}(G) = 3$.*

Proof sketch. We adapt a reduction from the 3-PARTITION problem to $\text{MFD}_{\mathbb{N}}$ on G , so that it maps 3-PARTITION to $\text{MFD}_{\mathbb{N}}$ instances of width 3. We are given a set of integers $A = \{a_1, \dots, a_{3q}\}$ for some $q \in \mathbb{N}$ and an integer B , such that $a_i \in (B/4, B/2)$. We want to find a partition of A into sets $S_1, \dots, S_q$ such that $\sum_{s \in S_i} s = B$. This implies that the sets must be of size 3.

The constructed $\text{MFD}_{\mathbb{N}}$ instance of width 3 can be seen in Figure 4.2. It has the following structure: it consists of two grid-like parts that are separated by a bridge and are displayed as the top and as the bottom components in Figure 4.2. The grids each contain diagonal edges of flow weight 1. By the large enough choice of a'_i and B' , every minimum flow decomposition must contain a path of weight 1 which saturates each such diagonal edge. After saturating them, we obtain a graph that can be Y-to-V contracted to the series-parallel instance of the original 3-PARTITION reduction in Figure 2.2. $\square$

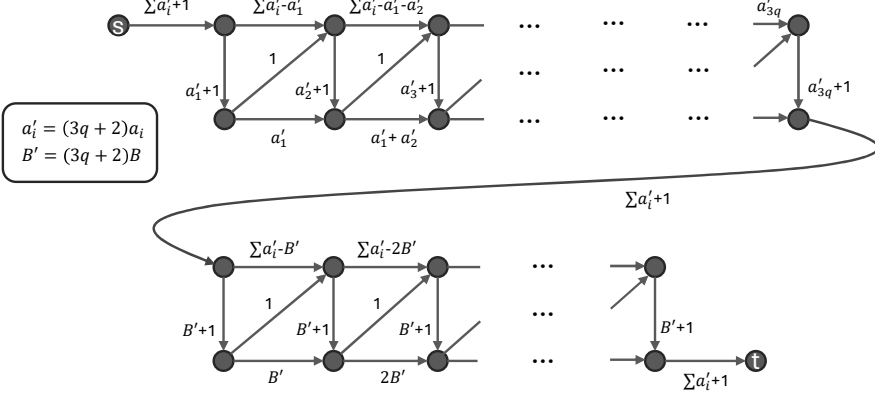


Figure 4.2: [Paper III, Figure 4] An $\text{MFD}_{\mathbb{N}}$ instance of width 3 for solving 3-PARTITION problem.

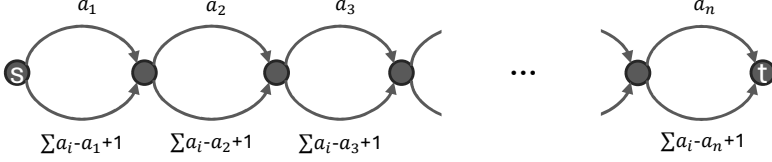


Figure 4.3: [Paper III, Figure 5] An $\text{MFD}_{\mathbb{N}}$ instance (G_A, X_A) of width 2 for solving GENERATING SET problem for a set A .

We refer to Paper III for a formal proof of Theorem 4.19. Note that this theorem also shows the fact that for any two constants $c, c' \geq 3$, there exists a DAG G with $\text{width}(G) \leq c$ and $\text{par-width}(G) \geq c'$.

Theorem 4.20 ([Paper III, Theorem 8]). *$\text{MFD}_{\mathbb{N}}$ on (G, X) is NP-hard when X is coded in binary, even when $\text{width}(G) = 2$.*

Proof sketch. We define a reduction from the GENERATING SET problem to an $\text{MFD}_{\mathbb{N}}$ instance (G_A, X_A) with $\text{width}(G) = 2$ in Figure 4.3. In the GENERATING SET problem, we are given a set $A = \{a_1, \dots, a_n\} \subseteq \mathbb{N}$ and an integer $k \in \mathbb{N}$, and we want to decide whether there is a set $Z = \{z_1, \dots, z_k\} \subseteq \mathbb{N}$, such that every $a \in A$ is the sum of some subset of the elements in Z . The reduction works in the following way. The bottom edges in Figure 4.3 each contain a large amount of flow, so that we must

use a path in the decomposition that only traverses bottom edges. Every other path traverses some of the top edges, and their weights will be one of the integers in Z . That is, when minimising k , we have $|Z| = k$ if and only if $\text{mfd}_{\mathbb{N}}(G, X_A) = k + 1$. $\square$

We refer to Paper III for a formal proof of Theorem 4.20. The GENERATING SET problem is known to be NP-hard [7] and the proof uses exponentially growing integers. Moreover, the best known complexity for GENERATING SET for unary coded integers is also quasi-polynomial and thus aligns with Theorem 4.16. This shows that $\text{MFD}_{\mathbb{N}}$ naturally generalises GENERATING SET. An FPT algorithm for $\text{MFD}_{\mathbb{N}}$ parameterised by $k = \text{mfd}_{\mathbb{N}}(G, X)$ running in time $2^{O(k)} \cdot \text{poly}(n, m, \log \|X\|)$ would, with the argument of Theorem 4.16, imply that GENERATING SET and $\text{MFD}_{\mathbb{N}}$ with constant parallel-width can be solved in polynomial time when the integers are coded in unary.

4.3.2 Solving the k -Flow Weight Assignment Problem

In this section, we discuss a restriction of $\text{MFD}_{\mathbb{N}}$ discussed by Kloster et al. [22] that is relevant for solving the problem (see Figure 4.4 for an example). It is used as a subroutine in the $\text{MFD}_{\mathbb{N}}$ FPT solver Toboggan [22].

Definition 4.21 (k -Flow Weight Assignment). Given a flow $X: E \rightarrow \mathbb{Y}$ on a DAG $G = (V, E)$ and a set of s - t paths $\{P_1, \dots, P_k\}$, the problem of finding an assignment of weights to the paths, such that they form a flow decomposition of X , is called *k -Flow Weight Assignment* (k -FWA). We write $k\text{-FWA}_{\mathbb{Y}}$ if we require the path weights to belong to $\mathbb{Y}$.

We can solve $k\text{-FWA}_{\mathbb{Y}}$ with a linear system of equations. We define it by $Lw = X$, where $X_j \in \mathbb{Y}$ is the flow weight of $X(e_j)$ and $L \in \{0, 1\}^{m \times k}$ is an indicator matrix with $L_{i,j} = 1$ if and only if $e_i \in P_j$. The variable w is the weight assignment to the paths. For a flow network (G, X) , we denote by $L_{\mathbb{Y}} = L_{\mathbb{Y}}(P_1, \dots, P_k) = \{w \in \mathbb{Y}^k \mid X = \sum_{j=1}^k P_j w_j\}$ the linear system corresponding to the paths $P_1, \dots, P_k$.

When solving $k\text{-FWA}_{\mathbb{Z}}$, we are given a linear system without lower or upper bounds to w , which is a system of linear Diophantine equations. It is well known that they can be solved in polynomial time [32].

In $k\text{-FWA}_{\mathbb{N}}$, we are given a linear system that contains the inequality $w \geq 0$. In fact, Kloster et al. [22] have shown that $k\text{-FWA}_{\mathbb{N}}$ is NP-hard. However, the proof is not restricted to the instances that occur in the subroutine in Toboggan, which motivates the following conjecture.

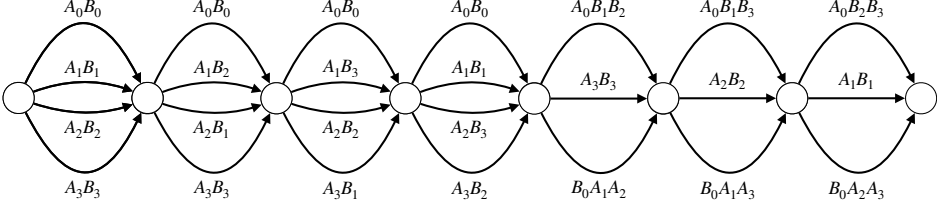


Figure 4.4: [Paper I, Figure 9] $\text{MFD}_{\mathbb{N}}$ instance (G, X) with paths A_i and B_i ($i \in \{0, 1, 2, 3\}$) labeled on every edge that the paths traverse. We assign to each path A_i a weight a_i , and to each path B_i a weight b_i , where we choose $a_0 = b_0 = 3$ and $a_i = 6^{2i} + 1$ and $b_i = 6^{2i+1} + 5$ for $i = 1, 2, 3$. Define the flow X on G as $X = \sum_{i=0}^3 a_i A_i + \sum_{i=0}^3 b_i B_i$. Note that these weights are a solution of $k\text{-FWA}_{\mathbb{N}}$ on (G, X) with the given paths A_i, B_i ($i \in \{0, 1, 2, 3\}$).

Conjecture 4.22 ([22]). *If $(P_1, \dots, P_k)$ are the paths of a minimum flow decomposition of X on a DAG G , then the linear system $L_{\mathbb{N}}(P_1, \dots, P_k)$ has full rank k .*

Vatinen et al. [37] proved that for $\mathbb{Y} = \mathbb{Q} \cap [0, \infty)$, the induced linear system is indeed of full rank k . As observed by Vatinen et al. [37], if the conjecture turned out to be true for natural numbers, the running time upper bound of Toboggan would improve, because the step of solving $k\text{-FWA}_{\mathbb{N}}$ as ILP could be replaced by solving it as linear program and checking whether the returned solution is integral.

We show that this conjecture is false using a counterexample. Consider the $k\text{-FWA}_{\mathbb{N}}$ instance in Figure 4.4 and the solution described therein. The instance contains another solution, namely the following weights: $a_0 = 5, b_0 = 1$ and $a_i = 6^{2i} + 2, b_i = 6^{2i+1} + 4$ for $i = 1, 2, 3$ (where each path A_i gets assigned the weight a_i and each B_i gets assigned b_i). Since the instance contains two distinct solutions of size 8, the rank of the corresponding linear system is smaller than 8.

It remains to show that $\text{mfd}_{\mathbb{N}}(G, X) = 8$ (i.e. the instance obtains no flow decomposition smaller than 8). We have verified this by running Toboggan [22] and by running the ILP solver by Dias et al. [10], both programs returning $\text{mfd}_{\mathbb{N}}(G, X) = 8$.

This can also be verified by a careful analysis of the instance. The values a_i and b_i increase exponentially for increasing i . In particular, we have

$$\begin{aligned} (a_0 + b_0) + \dots + (a_i + b_i) &< a_{i+1}, \\ (a_0 + b_0) + \dots + (a_i + b_i) + a_{i+1} &< b_{i+1}, \end{aligned}$$

for $i = 0, 1, 2$. We can use this property in the case by case analysis. Consider the set of all (edge) antichains $\mathcal{C}$ in the graph, and let it be ordered, such that C^0 is the left most antichain, C^1 is the the neighbouring antichain, and so on. The antichains C^0 to C^3 each contain four edges, and the antichains C^4 to C^6 each contain three edges. We can show that 8 is a lower bound to the number of paths crossing C^0 . We demonstrate the first case and omit the remaining ones. Intuitively, the permutations of the rapidly growing weights, together with the odd parity edges in C^4 to C^6 , force us to use many paths.

Case 1: Assume that the edge in C^0 with label A_3B_3 gets traversed by four weighted paths, then at least one other edge in C^0 is traversed by at least two weighted paths (resulting in at least eight total paths). Assume by contradiction, every other edge in C^0 is traversed by at most one weighted path. The path that traverses the edge labelled A_2B_2 in C^0 must then traverse the edge labelled A_3B_3 in C^1 , because all other weights in C^1 are smaller than $a_2 + b_2$. In addition, at least two of the paths traversing the edge in C^0 labelled A_3B_3 must traverse the edge in C^1 with the same label, because if only one path traversed the edge in C^1 labelled A_3B_3 , then it would be the only path of weight $\geq a_3$. But there need to be two such paths, since the labels A_3 and B_3 permute in later antichains. The at most other two paths traversing the edge in C^0 labelled A_3B_3 must traverse some other edge in C^1 . Thus, at most three paths cover the edges in C^1 labelled A_1B_2 and A_2B_1 . One of them has weight equal to $a_1 + b_2$ or $a_2 + b_1$ by the Pigeonhole principle. That is, we have at least four even parity paths (this path, and the three paths covering the first three edges of C^1 by themselves). Two of the potentially odd paths that traversed the edge labelled A_3B_3 in C^1 must traverse the edge with the same label in C^4 . We have at most one odd parity path and four even parity paths left to decompose the other two edges of C^4 , which are both odd. Thus, the parities do not match.

Cases 2-4: Assume that the edge in C^0 with label A_3B_3 is traversed by three, two, or one paths. We can show a contradiction or a lower bound of 8 paths traversing C^0 similarly to Case 1.

Chapter 5

ILP optimisations for practical flow decompositions

In the following, we improve solving $\text{MFD}_{\mathbb{N}}$ in the context of practical use cases. We first present an existent Integer Linear Program (ILP) model for $\text{MFD}_{\mathbb{N}}$ and for two practical variants of $\text{MFD}_{\mathbb{N}}$. We then optimise the ILP solver for $\text{MFD}_{\mathbb{N}}$ by performing pre-computations that reduce the branch-and-bound search space and the model dimension, and additionally we perform simple adjustments to the optimisations to apply them to the variants of $\text{MFD}_{\mathbb{N}}$.

We have decided to optimise the recent ILP-based solver for $\text{MFD}_{\mathbb{N}}$ by Dias et al. [10] for the following reasons. The model is flexible and can easily be adjusted to solve variations of $\text{MFD}_{\mathbb{N}}$, as described in the next paragraph. In addition, this ILP model is currently the state-of-the-art solver in terms of runtime. However, there are still instances in practice that cannot be solved feasibly. We experimentally show that the optimisations give a speedup factor of up to $34\times$ on the standard $\text{MFD}_{\mathbb{N}}$, and a speedup factor of even up to $219\times$ on one $\text{MFD}_{\mathbb{N}}$ variant.

In addition to the standard variant (which is just $\text{MFD}_{\mathbb{N}}$), we consider two more variants. Minimum *Inexact* Flow Decomposition ($\text{MIFD}_{\mathbb{N}}$), where we decompose any feasible flow following lower- and upper bound constraints, and Minimum Flow Decomposition *with Subpath Constraints* ($\text{MFDSC}_{\mathbb{N}}$) where we are given paths in the input that must appear in weighted path in the minimum flow decomposition. Afterwards, we define a new ILP model for the fourth variant Minimum Flow Decomposition *with given Weights* ($\text{MFDW}_{\mathbb{N}}$), where we are given a set of possible weights in the input, from which the paths' weights must come from. We show that this model can act as fast and accurate heuristic to $\text{MFD}_{\mathbb{N}}$.

5.1 ILP models of $\text{MFD}_{\mathbb{N}}$ and its variants

We first review the original ILP formulation by Dias et al. [10] for $\text{MFD}_{\mathbb{N}}$ and its two variations $\text{MIFD}_{\mathbb{N}}$ and $\text{MFDSC}_{\mathbb{N}}$. It decomposes the flow into a given parameter of k paths. The optimal solution can be found by an iteration starting from $k = 1$, until the ILP model becomes feasible. Note that this model does not contain an objective function.

$$X(u, v) = \sum_{i=1}^k \chi_i(u, v) \cdot w_i \quad \forall (u, v) \in E(G), \quad (5.1)$$

$$\sum_{u \in N^+(v)} \chi_i(v, u) = \sum_{u \in N^-(v)} \chi_i(u, v) \quad \forall v \in V(G), i \in [k], \quad (5.2)$$

$$\sum_{u \in N^+(s)} \chi_i(s, u) = 1 \quad \forall i \in [k], \quad (5.3)$$

$$w_i \in \mathbb{N} \quad \forall i \in [k], \quad (5.4)$$

$$\chi_i(u, v) \in \{0, 1\} \quad \forall (u, v) \in E(G), i \in [k]. \quad (5.5)$$

Equation 5.2 is the flow conservation constraint and Equation 5.3 enforces that the χ_i describe a path¹. We call the variables χ_i path indicators and equivalently consider them as characteristic functions: $\chi_i : E \rightarrow \{0, 1\}$ and $\chi_i(u, v) = 1$ if and only if the edge (u, v) is in the i -th path.

Remark 5.1. Equation 5.1 is not linear. However, Dias et al. [10] observed that it can be linearised with standard techniques. This involves introducing $k \cdot m$ additional variables $\phi_i(u, v)$ and additional constraints:

$$X(u, v) = \sum_{i=1}^k \phi_i(u, v) \quad \forall (u, v) \in E(G), \quad (5.6)$$

$$\phi_i(u, v) \leq \|X\| \cdot \chi_i(u, v) \quad \forall (u, v) \in E(G), i \in [k], \quad (5.7)$$

$$\phi_i(u, v) \leq w_i \quad \forall (u, v) \in E(G), i \in [k], \quad (5.8)$$

$$\phi_i(u, v) \geq w_i - (1 - \chi_i(u, v)) \cdot \|X\| \quad \forall (u, v) \in E(G), i \in [k], \quad (5.9)$$

$$\phi_i(u, v) \in \mathbb{N} \quad \forall (u, v) \in E(G), i \in [k]. \quad (5.10)$$

See the work of Dias et al. [10] for a proof. We omit these additional constraints in the future.

We consider the following variant of $\text{MFD}_{\mathbb{N}}$. Let $\mathcal{R}$ be a set of paths in G . We want to find a minimum flow decomposition, with the constraint

¹Recall that flows $X : E \rightarrow \mathbb{N}$ of value 1 are paths.

that every path in $\mathcal{R}$ is a subpath of at least one weighed path in the decomposition, and we call this problem Minimum Flow Decomposition with Subpath Constraints ($\text{MFDW}_{\mathbb{N}}$). Dias et al. [10] showed that it can be solved via the following additional constraints:

$$\sum_{(u,v) \in R_j} \chi_i(u, v) \geq |R_j| r_{ij} \quad \forall i \in [k], R_j \in \mathcal{R}, \quad (5.11)$$

$$\sum_{i=1}^k r_{ij} \geq 1 \quad \forall R_j \in \mathcal{R}, \quad (5.12)$$

$$r_{ij} \in \{0, 1\} \quad \forall i \in [k], R_j \in \mathcal{R}. \quad (5.13)$$

Moreover, we consider a third variant of $\text{MFD}_{\mathbb{N}}$, which describes noisy data in practical scenarios. Instead of decomposing a given flow, we are given an inexact flow network (G, L, R) with interval constraints $[L_e, R_e]$ for every edge $e \in E(G)$, and ask for a minimum flow decomposition that decomposes any flow X that satisfies $L_e \leq X(e) \leq R_e$ for all edges $e \in E(G)$. We call this variant Minimum Inexact Flow Decomposition ($\text{MIFD}_{\mathbb{N}}$). Dias et al. [10] gave a simple modification to the ILP model, by only changing Equation 5.1:

$$L_{uv} \leq \sum_{i=1}^k \chi_i(u, v) \cdot w_i \leq R_{uv} \quad \forall (u, v) \in E(G). \quad (5.14)$$

5.2 Dimensionality and search space reduction

In the following, we present an improvement to the ILP models in the form of pre-computed optimisations. We first show the optimisations for $\text{MFD}_{\mathbb{N}}$, and then explain our adaptation for the variants $\text{MFDSC}_{\mathbb{N}}$ and $\text{MIFD}_{\mathbb{N}}$. We benchmark the optimisations later in Section 5.4.

Search space reduction using safe paths. The goal is to present an optimisation that fixes as many path indicator variables $\chi_i(u, v)$ as possible. The core concept that we use is *solution safety*.

Definition 5.2. Let (G, X) be a network flow on a DAG G . We call a path P in G *safe* for X , if for every flow decomposition $(\mathcal{P}, w)$ of X , $P \subseteq Q$ for some $Q \in \mathcal{P}$. When X is clear from the context, we just call P a safe path.

Solution safety was first introduced by Tomescu et al. [36] and has since been used in several contexts. In natural sciences such as computational biology, optimisation problems often contain multiple optimal solutions, but among them, only one *biologically correct* solution. Thus, without additional information, it is not always possible to predict the correct solution. For example, solution safety has been used in protein sequence alignments in order to find protein regions with important functionality that are conserved throughout evolution [18].

Lemma 5.3 ([21]). *Let (G, X) be a flow network and let $P = (e_1, \dots, e_\ell)$ with $e_i = (u_i, u_{i+1})$ be a path in G . We define the excess flow of P by*

$$f_P := X(e_1) - \sum_{\substack{i=2, \dots, \ell \\ v \in N^+(u_i) \setminus \{u_{i+1}\}}} X(u_i, v).$$

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

Note that edges (as paths of length 1) are all safe. Khan et al. [21] characterised safe paths using excess flow, which gives us a polynomial time algorithm to find all safe paths of maximal lengths². First, find *any* flow decomposition, for example, of size $O(m)$. Then, for every weighted path in the decomposition, we find all subpaths of positive excess flow. This can be done in linear time using a two pointer algorithm: we move the right pointer if $f_P > 0$, and move the left pointer if $f_P \leq 0$. This finds all maximal safe paths contained in the weighted path of the decomposition. We refer to the work by Khan et al. [21] for a formal proof.

We now demonstrate that solution safety can speedup the ILP solver, by finding a set of safe paths that are pairwise independent. Two paths P and Q are called independent in G , if there is no s - t path in G that traverses both P and Q . Since they are pairwise independent, a set of size x of such paths must be decomposed by at least x different s - t paths in any flow decomposition. We can thus find an edge antichain $A = \{e_1, \dots, e_x\}$, find a safe path P_i containing e_i for every $e_i \in A$ and set $\chi_i(e) = 1$ for all $i = 1, \dots, |A|$, $e \in P_i$. Observe that such a set of safe paths is independent. It is left to decide which edge antichain to choose. We are interested in maximising the number of path indicator variables set to 1 and go for the following approach: for every edge e , find the length ℓ_e of a longest safe path containing e . We then find a maximum weighted edge antichain, that is, an edge antichain A that maximises $\sum_{e \in A} \ell_e$. See Figure 5.1 for an example.

²Maximal safe paths are paths that are no longer safe when extended to the left or to the right.

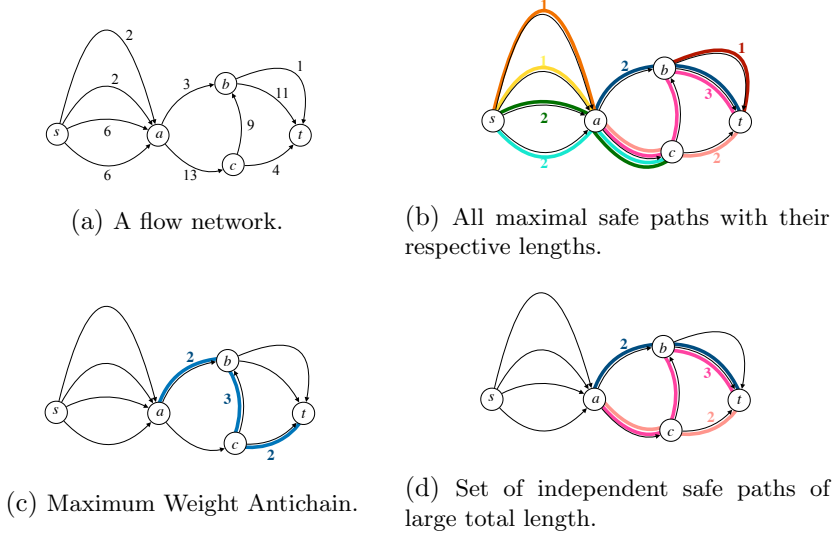


Figure 5.1: [Paper IV, Figure 2] Application of introducing new constraints in the ILP model by using safe paths. We start with the flow network displayed in Figure 5.1a. All maximal safe paths are computed and displayed in different colours with their respective lengths in Figure 5.1b. To every edge we assign as weight the maximum length of all safe paths crossing that edge. By computing a maximum weighted antichain (Figure 5.1c), we obtain a set of pairwise independent safe paths (Figure 5.1d).

A maximum weighted edge antichain can be found similarly to a maximum cardinality edge antichain: we reduce it to a min-flow problem, where the lower bound on the edges is given by their weights and the upper bounds are ∞ . We can find a minimum flow for example in runtime $O(m \cdot \sum_{e \in E(G)} \ell_e) \leq O(nm^2)$ [24].

Remark 5.4. The algorithm above is a heuristic and does not always find an optimal solution to the problem of maximising the total length of independent safe paths. See Figure 5.2 for an example. An advantage of the heuristic is its simplicity, which is the reason for our decision to implement it. We now present an algorithm that finds the optimal solution.

We define the following *dependency graph* $\mathcal{D} = (V(\mathcal{D}), E(\mathcal{D}))$: vertices represent safe paths in G and are weighted according to their lengths, and edges (p_1, p_2) are in $E(\mathcal{D})$ if safe path p_1 can reach safe path p_2 , that is, there exists a path P in G , which traverses p_1 and p_2 in this order. A maximum weighted node antichain on this graph is a maximum length set of independent safe paths. We refer to Paper IV, Appendix B for a proof.

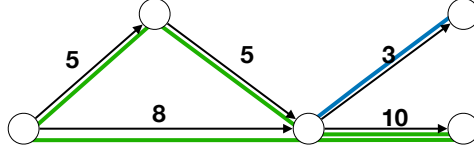


Figure 5.2: A flow network and its safe paths. This graph contains three maximal safe paths. One in green of length 3, one in green of length 2, and one in blue of length 1. All three safe paths are independent with a total length of 6. Our heuristic, which maximises the weights of an antichain, finds the two green independent safe paths of total length 5.

Y-to-V contractions and further optimisations. We use Y-to-V contractions to reduce the dimension of the ILP model and to reduce the number of constraints. We recall their definition from Chapter 3: a Y-to-V contraction is a contraction of an edge (u, v) where $\deg^+(u) = 1$ or $\deg^-(v) = 1$. This degree constraint marks redundant edges in the context of $\text{MFD}_{\mathbb{N}}$, as observed by Kloster et al. [22] (see Lemma 3.9). We employ the following bounds to the solution size k :

1. $k \geq |A|$, where A is the number of chosen independent safe paths,
2. $k \leq |\mathcal{P}|$, where $\mathcal{P}$ is the path set of a heuristic flow decomposition (that we use to compute all maximal safe paths).

Moreover, we remove symmetry by ordering the weights $w_i \leq w_{i+1}$ for all $i \in [k - 1]$.

Using safe paths to optimise variants. Every feasible solution to $\text{MFDSC}_{\mathbb{N}}$ is also a feasible solution to $\text{MFD}_{\mathbb{N}}$ without the subpath constraints. Thus, every path safe for $\text{MFD}_{\mathbb{N}}$ is also safe for $\text{MFDSC}_{\mathbb{N}}$ (assuming, the flow network can feasibly be decomposed with the given subpath constraints). We leave the open problem of finding an algorithm that finds all safe paths for a given $\text{MFDSC}_{\mathbb{N}}$ input.

For $\text{MIFD}_{\mathbb{N}}$, we adapt the excess flow to intervals. We show that paths of positive *inexact excess flow* are safe, and we leave the open problem of how to find all safe paths. The adaptation follows a conservative approach of leaking the highest possible flow away from the path, while starting with the smallest possible flow.

Definition 5.5 (Inexact excess flow [Paper IV, Lemma 4]). Let $P = (e_1, \dots, e_\ell)$ with $e_i = (u_i, u_{i+1})$ be a path in an $\text{MIFD}_{\mathbb{N}}$ instance (G, L, R) .

We define the *inexact excess flow* of P by the value

$$f_P = L_{e_1} - \sum_{\substack{i=2,\dots,\ell, \\ v \in N^+(u_i) \setminus \{u_{i+1}\}}} R_{u_i v}.$$

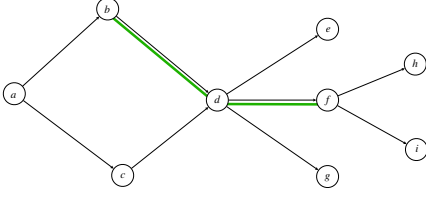
Lemma 5.6 ([Paper IV, Lemma 4]). *Assume that the inexact flow network (G, L, R) admits a feasible flow. A path P in G is safe if $f_P > 0$.*

Note that the converse statement is not necessarily true. For a path $P = (e_1, \dots, e_\ell)$ in G , it is possible that there is no feasible flow X with $X(e_1) = L_{e_1}$.

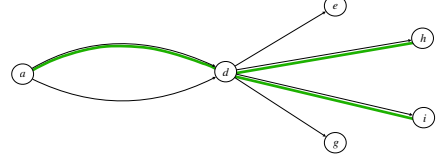
Y-to-V contractions for MFD_N variants. We now show how to adapt the Y-to-V contraction to the two variants. For MFDSC_N, we must translate the path constraints to the Y-to-V contracted graphs. We first perform the in-contractions and then the out-contractions. Recall that both of these actions partition the graph in out- and in-trees, as explained in Remark 3.8. We describe how to contract the paths for the in-contractions, the path contractions for the out-contractions follow equivalently.

Consider the out-tree partition of G . Each out-tree with k leaves l_j corresponds k edges from the root to each leaf (r, l_j) in the Y-to-V contracted graph $\hat{G}$. A path P in G traverses a sequence of out-trees $(T_1, \dots, T_\ell)$. We construct a new path P' by replacing the edges in P . For all $i = 2, \dots, \ell - 1$, the path passes through both the root node r_i and through a leaf node l_i of the out-tree T_i . We can replace all edges in P that come from T_i by the single edge (r_i, l_i) . The path starts at some node in T_1 and passes through a leaf node l_1 of T_1 . Since T_1 is an out-tree, any path that crosses P in G can only pass through the root node r_1 from T_1 . Thus, we can also replace all edges from T_1 by a single edge (r_1, l_1) . Finally, the path passes through the root node r_ℓ of the last tree T_ℓ , and might end before reaching a leaf node. Let P end at node $v \in T_\ell$, and let $L_{\ell,v}$ be the set of leaf nodes of T_ℓ that can be reached in G from v . We then replace all edges in P that are in T_ℓ by $|L_{\ell,v}|$ edges $(r_\ell, l_{\ell,j})$ for each $l_{\ell,j} \in L_{\ell,v}$. Note that P' is not a path when $|L_{\ell,v}| > 1$. It is a path of length $\ell - 1$, followed by $|L_{\ell,v}|$ many edges, from which exactly one has to be chosen, in order to cover P in the original graph. See Figure 5.3 for an example.

Let the set of such sub“path” constraints be $\hat{\mathcal{R}}$, and note that they are actually subgraphs. Let ℓ_j be the length of the longest path in the constraint. We adjust Constraints 5.11 and 5.12, and highlight the changes



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



(b) Y-to-V contracted graph with subpath constraint in green, which consists of a path of length 1 followed by two edges in the last out-tree. The ILP constraints enforce that exactly one of the two edges will be used.

Figure 5.3: [Paper IV, Figure 3] A DAG with one subpath constraint in green. The in-contractions partition the DAG in 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 and to end in two out-tree leaf nodes in (b).

in red:

$$\sum_{(u,v) \in R_j} \chi_i(u, v) \geq \textcolor{red}{\ell_j} r_{ij} \quad \forall i \in [k] \ R_j \in \hat{\mathcal{R}}, \quad (5.15)$$

$$\sum_{i=1}^k r_{ij} \geq 1 \quad \forall R_j \in \hat{\mathcal{R}}. \quad (5.16)$$

Lemma 5.7 ([Paper IV, Lemma 5]). *Constraints 5.15 and 5.16 hold in the Y-to-V contracted graph if and only if Constraints 5.11 and 5.12 hold in the original graph.*

In $\text{MIFD}_{\mathbb{N}}$, the intervals do not satisfy the flow conservation. Thus, some Y-to-V contractions will lose information and increase the set of feasible flows. Such new flows could potentially be decomposed into fewer paths. To circumvent this issue, we define the concept of *inexact flow conservation*.

Definition 5.8 (Generalisation of a definition in Paper IV, Section 4). Let (G, L, R) be an $\text{MIFD}_{\mathbb{N}}$ instance. We say that a node $v \in V(G)$ fulfils the

- *inexact in-flow conservation*, if

$$\sum_{u \in N^-(v)} L_{uv} \leq \sum_{u \in N^+(v)} L_{vu} \leq \sum_{u \in N^+(v)} R_{vu} \leq \sum_{u \in N^-(v)} R_{uv},$$

- *inexact out-flow conservation*, if

$$\sum_{u \in N^+(v)} L_{vu} \leq \sum_{u \in N^-(v)} L_{uv} \leq \sum_{u \in N^-(v)} R_{uv} \leq \sum_{u \in N^+(v)} R_{vu}.$$

If $\deg^-(v) = 1$ and v satisfies the inexact in-flow conservation (respectively, if $\deg^+(v) = 1$ and v satisfies the inexact out-flow conservation), we can safely contract the single incoming (respectively, outgoing) edge without increasing the set of feasible flows. Note that if (G, X) is a flow network, then (G, X, X) is an inexact flow (with exactly one feasible flow X) that satisfies the inexact in- and out-flow conservation on every node in $V(G) \setminus \{s, t\}$. The converse is not necessarily true, for example, in a path of length 2, where the intervals have length 2 and are equal in both edges of the path.

Lemma 5.9 ([Paper IV, Lemma 6]). *Let $(\hat{G}, \hat{L}, \hat{R})$ be an inexact flow instance obtained from Y -to- V contractions of (G, L, R) , where the contractions are only performed on edges (u, v) where $\deg^-(v) = 1$ and v satisfies the inexact in-flow conservation or $\deg^+(u) = 1$ and u satisfies the inexact out-flow conservation. Then every inexact flow decomposition of $(\hat{L}, \hat{R})$ uniquely determines an inexact flow decomposition of (L, R) and vice versa.*

5.3 An ILP solver for weighted $\text{MFD}_{\mathbb{N}}$

We now consider the last variant of $\text{MFD}_{\mathbb{N}}$ in this thesis, in which we are given a flow network (G, X) and a finite set of weights $W = \{w_1, \dots, w_\ell\} \subseteq \mathbb{N}$, and we ask for a flow decomposition of size k whose weights come only from W such that k is as small as possible. That is, we ask for a set of weighted paths $(\mathcal{P}, \tilde{w})$ such that $\sum_{P_i \in \mathcal{P}} P_i(e) \tilde{w}_i = X(e)$ for all $e \in E(G)$ with $\tilde{w} \in W^k$ and the size $|\mathcal{P}| = k$ is as small as possible.

Restricting the set of possible weights reduces the search space of possible flow decompositions and can potentially lead to faster ILP solvers. Moreover, in some applicable contexts, the weights of an optimal flow decomposition are already known. For example, Kloster et al. [22] have observed that the weights appear in the flow networks in the context of RNA sequence reconstructions.

If there exists a minimum flow decomposition using weights from W' , then we can solve $\text{MFDW}_{\mathbb{N}}$ using some weight set $W \supseteq W'$ and obtain an optimal $\text{MFD}_{\mathbb{N}}$ solution. For example, every flow X with $\|X\| \leq c$ for a constant c is decomposed by paths using weights from $[c]$.

Lemma 5.10 ([Paper IV, Corollary 9]). *$\text{MFDW}_{\mathbb{N}}$ is strongly NP-hard.*

Proof. Hartman et al. [19] showed that $\text{MFD}_{\mathbb{N}}$ is NP-hard even when the flow is restricted to consist of integers in $\{1, 2, 4\}$. $\square$

We now formulate an ILP model that decomposes a given flow X into a set of $|W|$ flows Y_w , such that $X = \sum_{w \in W} Y_w \cdot w$. Every flow Y_w can be decomposed into $\text{Val}(Y_w)$ paths, each of which are assigned the weight w .

$$\text{Minimise } \sum_{w \in W} \text{Val}(Y_w), \quad (5.17)$$

$$\text{Subject to} \quad (5.18)$$

$$X(e) = \sum_{w \in W} Y_w(e) \cdot w \quad \forall e \in E, \quad (5.19)$$

$$\sum_{u \in N^-(v)} Y_w(u, v) = \sum_{u \in N^+(v)} Y_w(v, u) \quad \forall v \in V \setminus \{s, t\}, w \in W, \quad (5.20)$$

$$Y_w(e) \in \mathbb{N} \quad \forall e \in E, w \in W. \quad (5.21)$$

The above constraints were defined in Paper IV, Section 5. There is no need for linearisation in this ILP model, which reduces the number of necessary variables and further reduces the runtime compared to the ILP for the standard $\text{MFD}_{\mathbb{N}}$ variant.

Lemma 5.11 ([Paper IV, Lemma 7]). *Constraints 5.17 to 5.21 solve $\text{MFDW}_{\mathbb{N}}$ optimally.*

$\text{MFDW}_{\mathbb{N}}$ can act as an $\text{MFD}_{\mathbb{N}}$ approximation by guessing the weights. For example, given (G, X) , we can choose $W = \{1, 2, 4, \dots, 2^{\lceil \log \|X\| \rceil}\}$, and obtain an approximation ratio of $O(\log \|X\|)$ for all DAGs. Recall that it is an open problem whether this can be done in polynomial time. Moreover, we can choose more sparse weight sets and obtain an approximation scheme, which acts as a trade off between running time and approximation ratio.

Lemma 5.12 ([Paper IV, Lemma 10]). *For an $\text{MFD}_{\mathbb{N}}$ instance (G, X) , solving $\text{MFDW}_{\mathbb{N}}$ with weights $\{2^j \mid j = 0, \dots, \lceil \log \|X\| \rceil\}$ optimally is a $\lceil \log \|f\| \rceil + 1$ -approximation of $\text{MFD}_{\mathbb{N}}$. More generally, solving $\text{MFDW}_{\mathbb{N}}$ of G with weights $W = \{(2^i)^j \mid j = 0, \dots, \lceil \log \|X\| / i \rceil\}$ for some $i \in \mathbb{N}$ is a $(2^i - 1)\lceil (\log \|X\| + 1) / i \rceil$ -factor approximation of $\text{MFD}_{\mathbb{N}}$.*

5.4 Experimental results

We have implemented the ILP solvers using the GUROBI Python API and implemented all optimisations using Python³. We compare the optimised

³The tool is available at <https://github.com/algbio/optimized-fd>.

ILPs (namely, MFD-optimized, MFDSC-optimized and MIFD-optimized) to the non-optimised ILPs for $\text{MFD}_{\mathbb{N}}$, $\text{MFDSC}_{\mathbb{N}}$ and $\text{MIFD}_{\mathbb{N}}$, and also compare them to the heuristics Catfish [33], CoasterHeuristic [39] and IFDSolver [38], respectively. For $\text{MFD}_{\mathbb{N}}$, we also run the ILP solver for $\text{MFDW}_{\mathbb{N}}$ using the weight set $W = \{2^i \mid i \leq \log \|X\|\} \cup \{X(e) \mid e \in E\}$. We call this solver using this weight set MFDW heuristic. This set of tools includes, next to our contributions, the state-of-the-art for optimal and heuristic solving in terms of runtime and approximation performance, respectively.

Datasets. We have decided to experiment with the datasets from previous articles:

1. For $\text{MFD}_{\mathbb{N}}$, we use the datasets created by Shao and Kingsford [33]. One dataset contains graphs created from human transcriptome using the quantification tool Salmon [28], and we use two datasets created from human and mouse transcriptomes using the Flux-Simulator [17]. The first dataset consists of a single sample file of the dataset from Shao and Kingsford [33], referred to as *SRR020730 Salmon*. For the remaining two datasets, we took the entire archive and refer to them as *Catfish Human* and *Catfish Mouse*, respectively. In the latter two datasets, we only considered instances whose Y-to-V contracted graph consisted of at least 50 vertices.
2. For $\text{MFDSC}_{\mathbb{N}}$, we use two datasets. An adaptation of *SRR020730 Salmon* with generated subpath constraints by Dias et al. [10] and *SRR30790 StringTie* by Khan et al. [21] which was created from human RNA-seq reads using the StringTie tool [29]. Both datasets only contain 4 subpath constraints per instance, due to the limitations of previously benchmarked tools.
3. For $\text{MIFD}_{\mathbb{N}}$, we simulate interval flows equivalently to Williams et al. [38], and repeat this process until feasible instances are generated.

Experimental setup. The experiments were performed on a server running Ubuntu on an AMD Ryzen Threadripper PRO 3975WX 32-core CPU with 512 GB RAM. We split every input file into several files each containing one instance, and use the GNU `time` tool to measure the runtime of each run. Every such run includes a linear scan over k to guess the optimal solution. Moreover, we use a 30 minute timelimit in GUROBI, and terminate once we found a solution or the ILP solver entered the timelimit.

Results. We show the runtime and the heuristics' approximation performance in Tables 5.1 to 5.3. We group the instances by the optimal solution

size that we have computed with our optimised solvers. We exclude the instances that it does not solve from the tables and note that only the heuristic algorithms, which do not find an optimal solution in general, are faster than our optimised solvers. In this way, we can compare the approximation performance of the heuristics.

Each solver has three columns, and we consider each dataset separately. For each group, we report the average running time for every solver in the column **Avg.** and the total running time of the group under **Total**. For the exact solvers we show the number of solved instances in the **#Solved** column and for the heuristic solvers (that is, the existent solvers for each variant and, in case of the standard $\text{MFD}_{\mathbb{N}}$ variant, the novel $\text{MFDW}_{\mathbb{N}}$ ILP solver) we show the difference of paths given by the heuristics and the optimal number of paths in the $\Delta(|\mathcal{P}|)$ column. We also show the average of the latter value in parantheses. The last line of every dataset shows the corresponding values for all groups combined.

Discussion. We illustrate the results for $\text{MFD}_{\mathbb{N}}$ in Table 5.1. MFD-optimized improves over the previous state-of-the-art MFD-original in all ranges. The scale of improvement increases with larger $\text{MFD}_{\mathbb{N}}$ size. We also observe that MFD-optimized solves more instances than MFD-original , which further increases the gap between the two solvers. Recall that we do not include instances in the runtime calculation that were unsolved due to not passing the timelimit. On *Catfish Human*, MFD-optimized is $21 - 31\times$ faster than MFD-original on average, and is $23\times$ faster than MFD-original on *Catfish Mouse* for $k \geq 21$ on average.

Among both heuristic solvers, *Catfish* finishes all instances in almost 0 seconds on average, whereas MFDW is slower, but still finishes in less than 1 second in all $\text{MFD}_{\mathbb{N}}$ size groups on average. However, MFDW approximates the optimal solution more accurately for larger k than *Catfish*. One possible reason might be that path weights are more likely to occur in the flow network as edge weight the more complex a flow network is. Recall that we chose W as all powers of two and all flow weights.

In Table 5.2, we show the results for the $\text{MFDSC}_{\mathbb{N}}$ problem. We observe an equivalent pattern as in $\text{MFD}_{\mathbb{N}}$. MFDSC-optimized improves on all ranges of k compared to MFDSC-original , and for $k \geq 21$, MFDSC-optimized remarkably runs $121\times$ faster than MFDSC-original . Moreover, MFDSC-optimized solves 2 more instances than MFDSC-original throughout both datasets.

In Table 5.3 we show the results for $\text{MIFD}_{\mathbb{N}}$ problem. We again observe that MIFD-optimized accelerates massively compared to MIFD-original . For example, on *Catfish Human*, MIFD-optimized is up to $219\times$ faster. More-

Table 5.1: [Paper IV, Table 1] Results for Problem $\text{MFD}_{\mathbb{N}}$. MFD-original denotes the original ILP for $\text{MFD}_{\mathbb{N}}$ from Dias et al. [10], MFD-optimized denotes our optimised ILP and MFDW the ILP solver for $\text{MFDW}_{\mathbb{N}}$. Runtimes are in seconds; a timeout of 30 minutes was used.

	min k	Catfish heuristic			MFDW heuristic			MFD-original			MFD-optimized		
		Avg.	Total	$\Delta(\mathcal{P})$	Avg.	Total	$\Delta(\mathcal{P})$	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

Table 5.2: [Paper IV, Table 2] Results for Problem $\text{MFDSC}_{\mathbb{N}}$. Runtimes are in seconds. A timeout of 30 minutes was used. MFDSC-original is the original ILP from from Dias et al. [10], and MFDSC-optimized is our optimised ILP solver. Note that all instances in the dataset *SRR30790 StringTie* have $k \leq 15$.

	min k	CoasterHeuristic			MFDSC-original			MFDSC-optimized		
		Avg.	Total	$\Delta(\mathcal{P})$	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

over, MFD-optimized runs less than 1 second in *all* solution size groups on average. In particular, it also solved all instances and can compare to IFDSolver heuristic, despite solving the problem optimally.

Finally, in order to measure each optimisation individually, we show quantitative statistics on their contributions in Table 5.4. We can see that the Y-to-V contractions remove more than half of the edges, even when the flow is inexact and not all contractions are performed. The proportion of path indicator variables $\chi_i(u, v)$ set to 1 by independent safe paths reaches almost 22% in one dataset, and is larger than 10% on most datasets. Moreover,

Table 5.3: [Paper IV, Table 3] Results for Problem MIFD_N. Runtimes are in seconds. A timeout of 30 minutes was used. MIFD-original is the original ILP from from Dias et al. [10], and MIFD-optimized is our optimised ILP. Note that the dataset *Catfish Human* has no instances with $k > 15$ and the dataset *Catfish Mouse* has no instances with $k > 20$.

	min k	IFDSolver heuristic			MIFD-original			MIFD-optimized		
		Avg.	Total	$\Delta(\mathcal{P})$	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

notably more than 91% of instances were solved optimally in *SRR020730 Salmon* for MFDSC_N using polynomial methods.

Table 5.4: [Paper IV, 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 found the optimal solution, using the heuristic solver and safety optimisations. The column p_{IND} shows the proportion of all path indicators $\chi_i(u, v)$ variables that were set to 1 as part of a safe path with respect to all path indicator variables of the contracted graph, where i is the index of the path and (u, v) is an edge. The last column $\#IND / \sum |P|$ shows the proportion of path indicator variables with respect to the sums of the solution's path lengths. Among assigning correct weights, the solution is found by precisely setting the indicator variables of the solution edges to 1.

	$ E $ original	$ E $ Y-to-V	$LB = UB$	p_{IND}	$\#IND / \sum P $
MFD_N					
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_N					
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_N					
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%

Chapter 6

Discussion and future directions

In this thesis we developed and analysed algorithms for decomposing flow networks by using structural invariants of directed graphs. Decomposing given objects is a natural task in graph theory, and the decompositions of flows specifically include various applications in the real world. We advanced the research in the lens of computational theory and practical solving. This work was published in four separate articles.

The theoretical research was motivated by the lack of approximation guarantees for commonly used heuristics. We argued that the DAG width, which has not previously been considered in the context of MFD , plays an important role in the decomposition of flows. For $\text{MFD}_{\mathbb{N}}$, we showed that flow decompositions correspond to a certain notion of directed minors. This allowed us to parameterise the problem by the parallel-width [8]. Moreover, we introduced the flow-width of a flow network (G, X) , and observed that, for certain flows X , it captures both the DAG width and the parallel-width of G .

Using these parameters, we obtained approximation upper bounds on certain classes of graphs. We introduced width-stable graphs as a natural generalisation of directed series-parallel graphs, and showed that we achieve an approximation ratio of $O(\log \|X\|)$ on them by using a parity-fixing algorithm. This is the case because the width remains a lower bound to the optimal solution throughout the decomposition, and we can use at most width many paths to fix the parity of any flow. More generally, the algorithm returns at most $\text{par-width}(G) \cdot O(\log \|X\|)$ many weighted paths on any graph, and we obtain the same approximation ratio on any instance with $\text{mfd}_{\mathbb{N}}(G, X) \geq \text{par-width}(G)$. Using this upper bound, we were able to provide a complete description of the complexity of $\text{MFD}_{\mathbb{N}}$, when it is parameterised by the width or the parallel-width. In contrast, we provide a class of instances, for which $\text{mfd}_{\mathbb{N}}(G, X) \ll \text{par-width}(G)$, where the parity-

fixing algorithm admits an approximation ratio of $\Omega(\|X\|)$ in the worst case.

Furthermore, we considered $\text{MFD}_{\mathbb{Z}}$, the version of the problem that allows negative integers, and designed an approximation algorithm for $\text{MCCD}_{\mathbb{Z}}$, which is a generalised problem of $\text{MFD}_{\mathbb{Z}}$. The algorithm fixes the parity of a circulation using a circulation of cost at most $\text{mccc}(G)$ (i.e., for $\text{MFD}_{\mathbb{Z}}$, using at most width many paths), and obtained an approximation factor of $O(\log \|X\|)$ on all instances. The advantage of $\text{MFD}_{\mathbb{Z}}$ compared to $\text{MFD}_{\mathbb{N}}$ is the fact that the width of the graph does not change, as edges do not get saturated.

From the practical perspective, we improved both exact and heuristic solving of $\text{MFD}_{\mathbb{N}}$. We optimised an existing ILP [10] by reducing the solver's search space and dimension, using (among other, smaller optimisations) the concept of solution safety, edge antichains and Y-to-V contractions. We also demonstrated that these optimisations can easily be adapted to variants of $\text{MFD}_{\mathbb{N}}$. We measured significant speedups, ranging from $34\times$ up to $219\times$ faster than the previous state-of-the-art ILP solver. We also defined a heuristic for $\text{MFD}_{\mathbb{N}}$ using ILP that guesses the weights, which is the most accurate heuristic on complicated instances and admits strong theoretical approximation guarantees. It is a large magnitude faster than the optimised ILP exact solver, but slower than the polynomial-runtime state-of-the-art heuristic.

Future work includes further development on the optimised ILP tool: improving the code base, making the tool more user friendly and making it compatible with free software ILP solvers. Our optimisations still leave room for improvement and further creative ideas.

There are many open questions left in the theory behind MFD. Is the problem in the class APX, or can it be shown that approximating it with any constant ratio is NP-hard? If this is the case, can we find a constant ratio approximation algorithm for all G where $\text{par-width}(G)$ is at most a constant? Can we extend/adjust the parity-fixing algorithm for $\text{MFD}_{\mathbb{N}}$, to obtain a $O(\log \|X\|)$ -ratio approximation algorithm for all instances, including $\text{mfd}_{\mathbb{N}}(G, X) \ll \text{par-width}(G)$?

References

- [1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice-Hall, Inc., 1993.
- [2] Jasmijn A. Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-Aware Assembly of Genomes from Mixed Samples Using Flow Variation Graphs. In *Proceedings of the 24th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 221–222. Springer, 2020.
- [3] 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.
- [4] Dietrich Braess. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.
- [5] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, 1999.
- [6] Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–376. SIAM, 2022.
- [7] Michael J Collins, David Kempe, Jared Saia, and Maxwell Young. Nonnegative integral subset representations of integer sets. *Information Processing Letters*, 101(3):129–133, 2007.
- [8] Argyrios Deligkas and Reshef Meir. Directed Graph Minors and Serial-Parallel Width. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 117,

- pages 44:1–44:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- [9] Fernando HC Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics*, 39(11):btad640, 2023.
 - [10] Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Fast, flexible, and exact minimum flow decompositions via ilp. In *Proceedings of the 26th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 230–245. Springer, 2022.
 - [11] RP Dilworth. A Decomposition Theorem for Partially Ordered Sets. *The Annals of Mathematics*, 51(1):161–166, 1950.
 - [12] David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992.
 - [13] Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.
 - [14] Marcelo Garlet Millani, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Efficient algorithms for measuring the funnel-likeness of dags. *Journal of Combinatorial Optimization*, 39:216–245, 2020.
 - [15] Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC Bioinformatics*, 20(1):1–14, 2019.
 - [16] Daniel Gibney, Sharma V Thankachan, and Srinivas Aluru. Feasibility of flow decomposition with subpath constraints in linear time. In *Proceedings of the 22nd International Workshop on Algorithms in Bioinformatics (WABI)*, pages 17:1–17:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
 - [17] 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.
 - [18] Andreas Grigorjew, Artur Gynter, Fernando HC Dias, Benjamin Buchfink, Hajk-Georg Drost, and Alexandru I Tomescu. Sensitive

- inference of alignment-safe intervals from biodiverse protein sequence clusters using emerald. *Genome Biology*, 24:168, 2023.
- [19] Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In *Proceedings of the IEEE INFOCOM 2012*, pages 828–836. IEEE, 2012.
- [20] Enver Kayaaslan, Thomas Lambert, Loris Marchal, and Bora Uçar. Scheduling series-parallel task graphs to minimize peak memory. *Theoretical Computer Science*, 707:1–23, 2018.
- [21] Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I Tomescu. Safety and completeness in flow decompositions for RNA assembly. In *Proceedings of the 26th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 177–192. Springer, 2022.
- [22] 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 *Proceedings of the 20th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, 2018.
- [23] Sven O Krumke and Christiane Zeck. Generalized max flow in series-parallel graphs. *Discrete Optimization*, 10(2):155–162, 2013.
- [24] Rolf H. Möhring. *Algorithmic Aspects of Comparability Graphs and Interval Graphs*, pages 41–101. Springer Netherlands, 1985.
- [25] Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. Parity Balancing Path Flow Decomposition and Routing. In *Proceedings of the 2015 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2015.
- [26] Nils Olsen, Natalia Kliewer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time-space network models. *Central European Journal of Operations Research*, 30(3):883–919, 2022.
- [27] James B Orlin. Max flows in $O(nm)$ time, or better. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 765–774, 2013.

- [28] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017.
- [29] 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.
- [30] Neil Robertson and Paul D Seymour. Graph minors I-XXIII. *Journal of Combinatorial Theory, Series B*, 1983-2010.
- [31] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2635. SIAM, 2019.
- [32] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.
- [33] 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.
- [34] 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.
- [35] Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. 14:1–10, 2013.
- [36] Alexandru I Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017.
- [37] Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe 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.

- [38] Lucia Williams, Gillian Reynolds, and Brendan Mumey. RNA transcript assembly using inexact flows. In *Proceedings of the 2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914. IEEE, 2019.
- [39] Lucia Williams, Alexandru I Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, 2022.

Paper I

Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams

Width Helps and Hinders Splitting Flows

In *Proceedings of the 30th Annual European Symposium on Algorithms (ESA 2022)*,

Leibniz International Proceedings in Informatics (LIPIcs), 2022, pages 31:1-31:14.

Copyright © Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams.

Width Helps and Hinders Splitting Flows

Manuel Cáceres  

Department of Computer Science,
University of Helsinki, Finland

Andreas Grigorjew  

Department of Computer Science,
University of Helsinki, Finland

Brendan Mumey 

School of Computing, Montana State University,
Bozeman, MT, USA

Alexandru I. Tomescu  

Department of Computer Science,
University of Helsinki, Finland

Massimo Cairo 

Department of Computer Science,
University of Helsinki, Finland

Shahbaz Khan  

Department of Computer Science and Engineering,
Indian Institute of Technology Roorkee, India

Romeo Rizzi  

Department of Computer Science,
University of Verona, Italy

Lucia Williams  

School of Computing, Montana State University,
Bozeman, MT, USA

Abstract

Minimum flow decomposition (MFD) is the NP-hard problem of finding a smallest decomposition of a network flow X on directed graph G into weighted source-to-sink paths whose superposition equals X . We focus on a common formulation of the problem where the path weights must be non-negative integers and also on a new variant where these weights can be negative. We show that, for acyclic graphs, considering the *width* of the graph (the minimum number of s - t paths needed to cover all of its edges) yields advances in our understanding of its approximability. For the non-negative version, we show that a popular heuristic is a $O(\log |X|)$ -approximation ($|X|$ being the total flow of X) on graphs satisfying two properties related to the width (satisfied by e.g., series-parallel graphs), and strengthen its worst-case approximation ratio from $\Omega(\sqrt{m})$ to $\Omega(m/\log m)$ for sparse graphs, where m is the number of edges in the graph. For the negative version, we give a $(\lceil \log \|X\| \rceil + 1)$ -approximation ($\|X\|$ being the maximum absolute value of X on any edge) using a power-of-two approach, combined with parity fixing arguments and a decomposition of unitary flows ($\|X\| \leq 1$) into at most width paths. We also disprove a conjecture about the linear independence of minimum (non-negative) flow decompositions posed by Kloster et al. [ALENEX 2018], but show that its useful implication (polynomial-time assignments of weights to a given set of paths to decompose a flow) holds for the negative version.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Network flows

Keywords and phrases Flow decomposition, approximation algorithms, graph width

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.31

Related Version *Full Version:* <https://arxiv.org/abs/2207.02136> [5]

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, SAFEBO), partially by the Academy of Finland (grants No. 322595, 328877), and partially by the National Science Foundation (NSF) (grants No. 1661530, 1759522).

1 Introduction

Minimum flow decomposition (MFD) is the problem of finding a smallest sized decomposition of a network flow X on directed graph $G = (V, E)$ into weighted source-to-sink paths whose superposition equals X . We focus on the case where path weights are restricted to be integers. It is a textbook result [1] that if G is acyclic (a DAG) a decomposition using no more than $m = |E|$ paths always exists. However, MFD is strongly NP-hard [25], even on DAGs, and



© Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams;
licensed under Creative Commons License CC-BY 4.0

30th Annual European Symposium on Algorithms (ESA 2022).

Editors: Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman; Article No. 31; pp. 31:1–31:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

even when the flow values come only from $\{1, 2, 4\}$ [12]. Recent work has shown that the problem is FPT in the size of the minimum decomposition [14] and that it can be formulated as an ILP of quadratic size [7].

While difficult to solve, MFD is a key step in many applications. For example, MFD on DAGs is used to reconstruct biological sequences such as RNA transcripts [18, 23, 11, 3, 22, 26] and viral strains [2]. MFD can also be used to model problems in networking [25, 12, 15] and transportation planning [16], although in some of these applications there may be cycles in the input. Despite the ubiquity of the MFD problem, the gap in our knowledge about the approximability of MFD is large, even when restricting to DAGs. It is known [12] that MFD (even on DAGs) is APX-hard (i.e., there is some $\epsilon > 0$ such that it is NP-hard to approximate within a $(1 + \epsilon)$ factor), so in particular, MFD does not admit a PTAS, unless $P=NP$. Furthermore, it can be approximated with a factor of $\lambda^{\log \|X\|} \log \|X\|$ [15], where λ is the length of the longest source-to-sink path and $\|X\|$ is the largest flow value in the network. In this work, we attempt to fill in some of the gaps between these results.

A natural lower bound for the size of an MFD of a DAG is the size of a *minimum path cover* of the set of edges with non-zero flow (i.e., the minimum number of paths such that every such edge appears in *at least* one path) – this size is called the *width* of the network. This trivially holds because every flow decomposition is also such a path cover. These two notions are analogies of the more standard notions of path cover and width of the *node set*. The node-variants are classical concepts, with algorithmic results dating back to Dilworth and Fulkerson [8, 10]. Despite this, the width has not been given any attention in the MFD problem, and in particular it has never been used in approximation algorithms. In this paper, we show that the width can play a key role both in the analysis of popular heuristics, and in obtaining the first approximation algorithm for a natural variant of MFD.

We start with a relaxation of MFD in which flow decomposition may also use negative integer weights on flow paths, rather than strictly positive weights as has traditionally been considered [25, 12, 14]. An important observation that we leverage for this variant (unlike the positive-only version) is that “the width does not increase” as flow paths are chosen and removed. Using this, we give a $(\lceil \log \|X\| \rceil + 1)$ approximation algorithm for this variant. To differentiate both versions, we use $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ throughout the paper. While $\text{MFD}_{\mathbb{Z}}$ is a natural version of the problem, to our knowledge it has not been previously considered in the MFD literature. However, it can also have natural applications, since by applying $\text{MFD}_{\mathbb{Z}}$ on the difference between two flows, one can minimally explain the *differences* between them, e.g. to explain the differences in RNA expression between two tissue samples with the fewest number of up/down regulated transcripts, which is often the goal of RNA sequencing experiments [21]. Our approximation follows a *power-of-two* approach where the weights of the paths chosen are (positive or negative) powers of two. More specifically, observe that if all flow values are even, then one can divide them by 2 and obtain a flow X with smaller $\|X\|$ whose decomposition can be transformed back into a decomposition of X . In order to obtain such an even flow, we prove a basic property that can be of independent interest: given any integer flow X , there exists a *unitary* flow (its values are 0, +1, or −1) Y , such that $X + Y$ is even on every edge (Lemma 5). In addition, given a unitary flow Y , we show that Y can be decomposed into k paths of weight +1 or −1, such that k is at most the width of the graph (Corollary 8). We obtain the $(\lceil \log \|X\| \rceil + 1)$ approximation ratio (Theorem 11) by iteratively removing the unitary flow, dividing all flow values by 2, and preprocessing the graph so that its width is a lower bound on the size of the $\text{MFD}_{\mathbb{Z}}$.

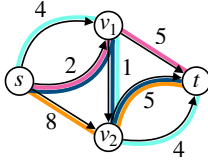
In Section 4 we consider connections between the width and a popular heuristic algorithm for $\text{MFD}_{\mathbb{N}}$ which we call *greedy-weight*¹ [25], which builds a flow decomposition by successively choosing the path that can carry the largest flow. Greedy-weight is commonly used in applications (see e.g., [23, 2, 18] among many), and it seems to be mentioned in nearly every publication addressing flow decomposition. First, on sparse graphs we improve (i.e., increase) the worst-case lower bound for the greedy-weight approximation factor from $\Omega(\sqrt{m})$ [12] to $\Omega(m/\log m)$, showing for the first time that greedy-weight can be exponentially worse than the optimum. For this we use a class of sparse graphs where the optimum flow decomposition has size $O(\log m)$ whereas the greedy-weight algorithm returns a solution of size $\Omega(m)$, only a constant factor away from the trivial decomposition. The key to this new bound is to design an input where the width increases exponentially when a path is greedily removed. We also show that the same bound also holds for other greedy heuristics choosing instead the longest or shortest paths. Second, we show that if the input satisfies the properties that its width does not increase as source-to-sink paths are removed (Property 15) and that it is possible to remove a path of large weight (Property 16), then greedy-weight is a $O(\log |X|)$ -approximation, where $|X|$ is the flow value (i.e., total flow out of s). A notable class of graphs satisfying these properties is the class of *series-parallel graphs*; see [9, 24] for fast recognition algorithms and pointers to other NP-hard problems that are easier on this class of graphs. Series-parallel graphs are also of great interest for network flow problems (see, e.g., [13, 4]).

Finally, in Section 5 we consider a closely related problem, called *k-Flow Weight Assignment* [14]. In addition to the flow X , in this problem we are also given a set of k paths, and we need to decide if there is an assignment of weights to the paths such that they form a decomposition of X . If the weights belong to $\mathbb{N}$, this was shown to be NP-complete in [14]. In this work, we first observe that in the same way that allowing negative integer weights simplifies the approximability of MFD, allowing weights to belong to $\mathbb{Z}$ fully changes the complexity of the k -Flow Assignment Problem, making it polynomial. This is due to the fact that the linear system defined by the given paths loses its only inequality of restricting the weights to positive integers. It thus transforms an ILP to a system of linear Diophantine equations, which can be solved in polynomial time (see e.g. [19]). Second, we consider a conjecture from [14] stating that if the weights belong to $\mathbb{N}$, and k is the size of a $\text{MFD}_{\mathbb{N}}$ for X , then the problem admits a unique solution (i.e., a unique assignment of weights to the given paths). If true, this would speed up the FPT algorithm of [14] for $\text{MFD}_{\mathbb{N}}$, because a step solving an ILP could be executed by solving a standard linear program returning a rational solution and checking if the (supposedly unique) solution to this system is integer. Moreover, the same conjecture (with the same implication) was also a motivation behind the greedy algorithm of [20] for $\text{MFD}_{\mathbb{N}}$. In this paper, we disprove the conjecture of [14], further corroborating the gap between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$.

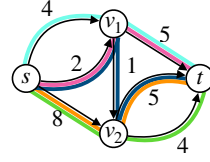
2 Preliminaries

We are given a directed graph $G = (V, E)$. Without loss of generality, we assume a unique source s and a unique sink t with no in-edges and no out-edges respectively; otherwise, the graph can be converted to such a graph by adding a pseudo source and sink and connecting them to all sources and sinks respectively with appropriately weighted edges. We also assume

¹ Previous work has consistently referred to this algorithm as *greedy-width*. To avoid confusion with the width of the graph, we introduce the name *greedy-weight* in this work.



(a) If negative weights are allowed, the four paths decompose the flow with weights 4, 5, 8, and -3 (dark blue).



(b) With positive weights only, five paths are needed, since the edge (v_1, v_2) must be decomposed by a weight 1 path, leaving 4 edges that must be covered separately. The paths shown are one such decomposition.

■ **Figure 1** A positive flow admitting a decomposition into four paths only if negative weights are allowed.

that every node is on some s - t path. We use n and m to denote the number of nodes and edges of G , respectively. Additionally, we assume that G is a DAG throughout the paper. While the problem is also studied for graphs with cycles (see, e.g., [25, 12]), the task is still to decompose into simple paths, and so our inapproximability result on DAGs also applies for graphs with cycles. We call functions $X: E \rightarrow \mathbb{Y}$ *pseudo-flows*, where $\mathbb{Y}$ is some set of allowed flow values (numbers). We treat pseudo-flows as vectors over E and use the notation $X + Y$ and aX to denote the (element-wise) sum of pseudo-flows and multiplication by a scalar, respectively. The numbers 0 and 1 also denote (depending on context) pseudo-flows that are 0 (resp. 1) everywhere. We write $X \leq Y$ (and similarly $<$) to mean $X(u, v) \leq Y(u, v)$ for every $(u, v) \in E$.

Given G , a *flow* is a pseudo-flow satisfying conservation of flow (incoming flow equal to outgoing flow) on internal nodes $V \setminus \{s, t\}$. It is known that the sum of two flows $X + Y$, the multiplication of a flow with a scalar aX , and the empty flow 0 are themselves flows. Let $|X|$ denote the total flow out of s (or into t) for flow X . Given an s - t path P , let P also denote the flow defined by setting 1 to every edge in P and 0 elsewhere. With these definitions, we are ready to formally define MFD.

► **Definition 1.** Given a flow X , a flow decomposition of (G, X) of size k is a family of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ with weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $X = w_1 P_1 + \dots + w_k P_k$.

► **Definition 2.** Given a flow X , let $\text{mfd}_{\mathbb{Y}}(G, X)$ be the smallest size of a flow decomposition of (G, X) with weights in $\mathbb{Y}$.

We omit $\mathbb{Y}$ if it is clear from the context. We call the problem of producing a flow decomposition of (G, X) of minimum size the *minimum flow decomposition (MFD) problem*. In this paper, we study two integer versions of the problem, $\text{MFD}_{\mathbb{N}}$ ($0 \in \mathbb{N}$) and $\text{MFD}_{\mathbb{Z}}$. Note that the reduction showing $\text{MFD}_{\mathbb{N}}$ to be strongly NP-hard from [25] also holds for $\text{MFD}_{\mathbb{Z}}$. However, a positive flow may admit a decomposition using fewer paths if negative weights are allowed, as shown in Figure 1. We explore further differences between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ in Sections 3 and 5.

Let $\|X\| = \max_{(u,v) \in E} |X(u, v)|$ denote the infinity norm on flows. In particular, notice that if $\mathbb{Y} \subseteq \mathbb{Z}$, then $\|X\| \leq 1$ means that $X(u, v) \in \{0, \pm 1\}$ for every $(u, v) \in E$. Let $X \equiv_2 Y$ if X and Y have the same parity everywhere, i.e., for every $(u, v) \in E$, we have that $X(u, v)$ is odd iff $Y(u, v)$ is odd.

► **Definition 3.** Given $S \subseteq E$, we define $\text{width}_S(G)$ as the minimum number of s - t paths in G needed to cover all edges of S . If $S = E$ we just write $\text{width}(G)$.

Just like its more common node variant, $\text{width}(G)$ can be computed in $O(mn)$ time. As described by, e.g., [1, 6], this is done by reduction to a min-flow instance with demand one on every edge; the minimum flow of this instance is $\text{width}(G)$, and the flow can be found by reduction to a max-flow instance. Moreover, the problem can be relaxed to only require the coverage of $S \subseteq E$ and solved in the same running time by setting the demands only on the edges of S .

► **Lemma 4** ([1, 17]). *Let $G = (V, E)$ be a DAG, and $S \subseteq E$. A flow $C : E \rightarrow \mathbb{N}$ can be computed in $O(mn)$ time, such that $C(e) \geq 1$ for all $e \in S$ and $|C| = \text{width}_S(G)$.*

The flow C with total flow $\text{width}_S(G)$ suffices in this paper, and we do not need to calculate a path cover achieving that minimum. However, we note that it can be directly computed given the flow C . We can think of this path cover as a flow decomposition of C into $\text{width}_S(G)$ weight-one paths, which can be found by greedily removing such paths from C until it is completely decomposed. Since each path has no more than $n - 1$ edges and since $\text{width}_S(G) \leq m$, the overall runtime of finding the path cover is $O(mn)$.

3 Width helps solve $\text{MFD}_{\mathbb{Z}}$

The idea behind our approximation algorithm for $\text{MFD}_{\mathbb{Z}}$ is that a flow $X : E \rightarrow \mathbb{Z}$ on DAG G can always be decomposed into $(\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G)$ paths. We show this using two key facts: first, that X can be decomposed into $(\lceil \log \|X\| \rceil + 1)$ flows with a particular structure, and, second, that each of these flows can be decomposed into $\text{width}(G)$ paths. A key step in proving both these facts is a subroutine which, given an input flow X , finds another flow Y with only values from $\{0, \pm 1\}$ (a *unitary* flow) that matches the parity of X on all the edges. Intuitively, given an input flow X , such a unitary flow Y can be added to X to “fix” its odd edges to be even, with only a small change to X .

► **Lemma 5.** *For any flow $X : E \rightarrow \mathbb{Z}$ on $G = (V, E)$, there exists a flow $Y : E \rightarrow \mathbb{Z}$ such that $X \equiv_2 Y$ and $\|Y\| \leq 1$.*

Proof. Consider the undirected graph $G_{\text{odd}} = (V, E_{\text{odd}})$ where $E_{\text{odd}} = \{\{u, v\} \mid (u, v) \in E \text{ and } X(u, v) \text{ is odd}\}$.

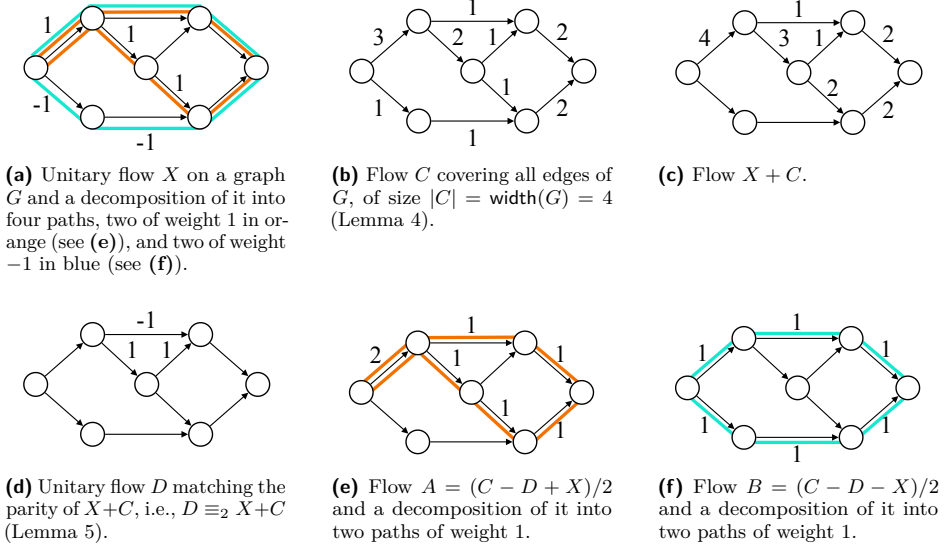
Notice that every node of G_{odd} , except possibly s and t , has even degree, due to conservation of flow. Thus, G_{odd} can be written as the edge-disjoint union of cycles and s - t paths. Assign an arbitrary orientation to each cycle and s - t path, and let E_{odd}^+ be the set of edges oriented in this way. Define

$$Y(u, v) = \begin{cases} +1 & \text{if } (u, v) \in E_{\text{odd}}^+ \\ -1 & \text{if } (v, u) \in E_{\text{odd}}^+ \\ 0 & \text{if } \{u, v\} \notin E_{\text{odd}} \end{cases}$$

Notice that Y is a flow decomposed as a sum of flows, each along one of the edge-disjoint cycles and s - t paths. Moreover, $X \equiv_2 Y$ and $\|Y\| \leq 1$ by construction. ◀

Repeatedly applying Lemma 5 and dividing the resulting even flow by 2, we obtain the first key ingredient of the approach (proof in [5, Appendix B]).

► **Corollary 6.** *Any (non-zero) flow $X : E \rightarrow \mathbb{Z}$ can be written as $X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i$, where $Y_i : E \rightarrow \mathbb{Z}$ is a flow with $\|Y_i\| \leq 1$ for all i .*



■ **Figure 2** Example of Lemma 7 and Corollary 8 applied to a unitary flow X on a graph G (for clarity, 0 flow values are not shown). Positive flows A and B can be constructed so that $|A| + |B| \leq \text{width}(G)$ holds. Flows A and B can be trivially decomposed into $|A|$ and $|B|$ paths, respectively. We obtain a decomposition of X by taking the paths of A with weight 1 and the paths of B with weight -1 .

The following result is the second key ingredient of our approach. It guarantees (together with Corollary 8) that any unitary flow can be decomposed into at most $\text{width}(G)$ paths of weight ± 1 (see Figure 2 for an example). This is by no means obvious since, among other problems, a unitary flow may contain positive and negative values which merge and cancel each other out (as in Figure 2a). The proof is based on another application of Lemma 5, along with some algebra on flows.

► **Lemma 7.** *For any flow $X : E \rightarrow \mathbb{Z}$, $\|X\| \leq 1$, there exist flows $A, B : E \rightarrow \mathbb{Z}$ such that:*

1. $A, B \geq 0$
2. $X = A - B$
3. $|A| + |B| \leq \text{width}(G)$

Proof. Take C such that $C \geq 1$ and $|C| = \text{width}(G)$, according to Lemma 4. Take D such that $D \equiv_2 X + C$ and $\|D\| \leq 1$, according to Lemma 5. Also, assume $|D| \geq 0$ without loss of generality (otherwise, take $-D$, which satisfies the same properties).

Since $D \equiv_2 X + C$, we have $C - D \pm X \equiv_2 0$. So we can take $A = (C - D + X)/2$ and $B = (C - D - X)/2$.

1. Notice that $C - D \pm X \geq C - 2$ since $\|D\|, \|X\| \leq 1$. So, $C - D \pm X \geq -1$, since $C \geq 1$. But $C - D \pm X \equiv_2 0$ so $C - D \pm X \geq 0$, whence $A, B \geq 0$.
2. $A - B = \frac{C-D+X}{2} - \frac{C-D-X}{2} = X$.
3. $|A| + |B| = |A + B| = \left| \frac{C-D+X}{2} + \frac{C-D-X}{2} \right| = |C - D| = |C| - |D| \leq |C|$ since $|D| \geq 0$, and $|C| = \text{width}(G)$. ◀

► **Corollary 8.** *For any flow $X : E \rightarrow \mathbb{Z}$ with $\|X\| \leq 1$, there exist paths $P_1, \dots, P_k$ with $k \leq \text{width}(G)$ such that $X = P_1 + \dots + P_\ell - P_{\ell+1} - \dots - P_k$ (for some $0 \leq \ell \leq k$).*

Proof. Take A, B according to Lemma 7, with $A, B \geq 0$, $X = A - B$ and $|A| + |B| \leq \text{width}(G)$. Since $A, B \geq 0$, there exist paths $P_1, \dots, P_{|A|+|B|}$ such that $A = P_1 + \dots + P_{|A|}$ and $B = P_{|A|+1} + \dots + P_{|A|+|B|}$ [1]. Since $X = A - B$, we can write $X = P_1 + \dots + P_{|A|} - P_{|A|+1} - \dots - P_{|A|+|B|}$. Finally, recall that $|A| + |B| \leq \text{width}(G)$, concluding the proof. ◀

Finally, expressing any flow as a sum of at most $\lceil \log \|X\| \rceil + 1$ unitary flows (Corollary 6), and decomposing each unitary flow into at most $\text{width}(G)$ positive or negative paths (Corollary 8), we can decompose the flow into at most $\lceil \log \|X\| \rceil + 1$ paths whose weight are positive and negative powers of two.

► **Theorem 9.** *Given a DAG $G = (V, E)$, for any flow $X: E \rightarrow \mathbb{Z}$, there exist paths $P_1, \dots, P_k$ and weights $\{w_1, \dots, w_k\} \subseteq \{\pm 2^i \mid i \in \mathbb{N}\}$, with $k \leq (\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G)$ such that $X = w_1 P_1 + \dots + w_k P_k$.*

Proof. Combine Corollaries 6 and 8, getting

$$X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot (P_{1_i} + \dots + P_{\ell_i} - P_{\ell_i+1} - \dots - P_{k_i})$$

where $k_i \leq \text{width}(G)$. ◀

The proof of Theorem 9 suggests a straightforward algorithm for $\text{MFD}_{\mathbb{Z}}$, which we detail in Algorithm 2 and describe at a high level here. First, iteratively decompose X , yielding $\log \lceil \|X\| \rceil + 1$ unitary flows. Then use Lemma 7 to decompose each into $\text{width}(G)$ paths. However, as we explained at the beginning of this section, $\text{width}(G)$ is not a lower bound on $\text{MFD}_{\mathbb{Z}}$, and thus this approach does not directly derives an approximation. To overcome this issue, we instead find a flow decomposition of a spanning subgraph G' of G whose width lower bounds $\text{mfd}_{\mathbb{Z}}(G, X)$. Namely, we first find a minimum path cover flow in G of the subset S of edges with non-zero flow in $O(mn)$ time (according to Lemma 4), and then remove from G any edge not covered by the flow, obtaining G' . By construction, the size of this path cover is a lower bound of $\text{mfd}_{\mathbb{Z}}(G, X)$. Moreover, the size of this path cover is exactly $\text{width}(G')$, since every path cover of G' is also a path cover of S in G .

To prove the correctness of Algorithm 2, we first define a subroutine implementing Lemma 5.

► **Lemma 10.** *Algorithm 1 returns a unitary flow from input flow Y such that $X \equiv_2 Y$, as in Lemma 5, in $O(m)$ time.*

Proof. The correctness of the algorithm is given by Lemma 5. Finally, the first 3 subroutines as well as the entire **for-loop** takes $O(m)$ time. ◀

► **Theorem 11.** *Algorithm 2 is a $\log \lceil \|X\| \rceil + 1$ -approximation for $\text{MFD}_{\mathbb{Z}}$ with runtime $O(m(\log \|X\| \cdot \text{mfd}_{\mathbb{Z}}(G, X) + n)) = O(m^2 \log \|X\|)$.*

Proof. By Theorem 9 and our previous discussion, Algorithm 2 returns a flow decomposition for X with no more than $(\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G') = (\lceil \log \|X\| \rceil + 1) \cdot \text{mfd}_{\mathbb{Z}}(G, X)$ paths. We analyze the runtime line by line. Lines 2 and 5 take $O(mn)$ time by Lemma 4. The call to Algorithm 1 on line 6 takes $O(m)$ time by Lemma 10, and checking the flow of D and flipping signs (if necessary) also takes $O(m)$ time. By Corollary 6, the while loop on line 8 executes at most $\log \lceil \|X\| \rceil + 1$ times, meaning that the entire execution takes $O(m \log \|X\|)$ time since line 9 takes $O(m)$ time by Lemma 10. Since there are at most $\log \lceil \|X\| \rceil + 1$ Y_i 's, the for loop on line 14 executes at most $\log \lceil \|X\| \rceil + 1$ times. Each

■ **Algorithm 1** Produces a unitary flow Y from input flow X such that $X \equiv_2 Y$, as in Lemma 5.

```

1: procedure UNITARY( $G, X$ )
2:    $E_{\text{odd}} \leftarrow$  odd edges of  $G$ , undirected
3:    $C \leftarrow$  a decomposition of  $G_{\text{odd}} = (V, E_{\text{odd}})$  into cycles, oriented arbitrarily
4:    $E_{\text{odd}}^+ \leftarrow$  directed edges of  $C$ 
5:   for  $(u, v) \in E$  do
6:     if  $(u, v) \in E_{\text{odd}}^+$  then
7:        $Y(u, v) \leftarrow +1$ 
8:     else if  $(v, u) \in E_{\text{odd}}^+$  then
9:        $Y(u, v) \leftarrow -1$ 
10:    else
11:       $Y(u, v) \leftarrow 0$ 
12:    end if
13:  end for
14:  return  $Y$ 
15: end procedure

```

execution of the for-loop naively finds $\text{width}(G')$ paths, each of which can be found in $O(m)$ time, so the whole loop takes $O(\log \|X\| \cdot \text{width}(G') \cdot m)$ time. Thus, the overall runtime is $O(m \log \|X\| \cdot \text{width}(G') + n) = O(m \log \|X\| \cdot \text{mfd}_{\mathbb{Z}}(G, X) + n)$. ◀

A theorem analogous to Theorem 9 for $\text{MFD}_{\mathbb{N}}$ is desirable, but cannot be achieved directly with the previous methods. Lemma 5 makes use of negative weights, and yields positive weights only if the flow graph solely consists of s - t paths. However, the approach can be adapted for $\text{MFD}_{\mathbb{N}}$ if the input flows are stable (Property 15), and if it is possible to “fix” the odd flows to be even with only $\text{width}(G)$ paths, which we leave as an open question.

4 Width matters for greedy approaches

Since the difference of two flows is still a flow, it is very natural to consider successively removing the most obvious type of flow – that is to say, paths – as an algorithmic strategy for $\text{MFD}_{\mathbb{N}}$. Indeed, the particular greedy path removal strategy of finding the *heaviest path* (*greedy-weight*) is commonly used as a heuristic in applications (e.g., [18, 2, 23, 12]) and it seems to be mentioned in nearly every paper addressing flow decomposition. More formally, a path P is said to *carry* flow p if $X(u, v) \geq p$ for all edges (u, v) of P . The *heaviest path* is an s - t path carrying the largest flow. Such a path can be easily found in linear time in the size of the DAG by dynamic programming (see, e.g., [25]).

4.1 Width hinders greedy on $\text{MFD}_{\mathbb{N}}$

We define a family of $\text{MFD}_{\mathbb{N}}$ instances $(G_{\ell}, X_{\ell, B})$, depending on two parameters $\ell \in \mathbb{N} \setminus \{0\}$ and $B \in \mathbb{N}$. The family is defined recursively on ℓ . The base case $(G_1, X_{1, B})$ for $\ell = 1$ is shown in Figure 3a. For $\ell > 1$, we build $(G_{\ell}, X_{\ell, B})$ from two disjoint copies of $(G_{\ell-1}, X_{\ell-1, B})$, by adding 5 extra edges and flow values as shown in Figure 3b. We call the edge connecting the two copies of $G_{\ell-1}$ a *central edge*. Edges whose flow value depends on B are called *backbone* edges, and they form a s - t path. Choosing $B = 2^{\ell+1}$, we show that the flow $X_{\ell, 2^{\ell+1}}$ can be decomposed using a number of paths linear in ℓ , thanks to the heavy backbone

■ **Algorithm 2** Finds the flow decomposition of Theorem 9.

```

1: procedure PATH-DECOMPOSITION( $G, X$ )
2:   Compute a minimum path cover flow of  $\{(u, v) \in E \mid X(u, v) \neq 0\}$  ▷ Lemma 4
3:   Remove from  $G$  any edge not covered by this path cover to obtain  $G'$ 
4:    $\mathcal{P} \leftarrow \square, w \leftarrow \square$  ▷ length-zero vectors
5:    $C \leftarrow$  flow of value  $\text{width}(G'), C \geq 1$  ▷ Lemma 4
6:    $D \leftarrow \text{UNITARY}(G', C)$ ; if  $|D| < 0$  set  $D = -D$  ▷ Algorithm 1
7:    $i \leftarrow 0$ 
8:   while  $\|X\| > 1$  do
9:      $Y_i \leftarrow \text{UNITARY}(G', X)$  ▷ Algorithm 1
10:     $X \leftarrow (X - Y_i)/2$ 
11:     $i \leftarrow i + 1$ 
12:  end while
13:   $Y_i \leftarrow X$ 
14:  for  $j \in \{0, \dots, i\}$  s.t.  $Y_j \neq 0$  do
15:     $A \leftarrow C - D + Y_j, B \leftarrow C - D - Y_j$ 
16:    Naively decompose  $A$  into  $|A|$  paths and  $B$  into  $|B|$  paths; concatenate  $A, B$  to  $\mathcal{P}$ 
17:    Concatenate  $|A|$  copies of  $2^j$  and  $|B|$  copies of  $-2^j$  to  $w$ 
18:  end for
19:  return  $(\mathcal{P}, w)$ 
20: end procedure

```

edges, whereas the greedy-weight algorithm fully saturates the central edges with its first path and is left with a remaining flow requiring $2^{\ell+1}$ paths to be decomposed (proofs in [5, Appendix B]).

► **Lemma 12.** *Let G_ℓ with flow $X_{\ell, 2^{\ell+1}}$ be constructed as described before. Greedy-weight uses $1 + 2^{\ell+1}$ paths to decompose $X_{\ell, 2^{\ell+1}}$.*

► **Lemma 13.** *Let G_ℓ with flow $X_{\ell, 2^{\ell+1}}$ be constructed as described before. It is possible to decompose $X_{\ell, 2^{\ell+1}}$ using $2\ell + 2$ paths.*

► **Theorem 14.** *The approximation ratio for greedy-weight on $\text{MFD}_{\mathbb{N}}$ is $\Omega(m/\log m)$ for sparse graphs, in the worst case.*

Proof. By Lemmas 12 and 13, greedy-weight uses $\Theta(2^\ell)$ paths to decompose the flow $X_{\ell, 2^{\ell+1}}$ described above, whereas it is possible to decompose the flow with only $\Theta(\ell)$ paths. It can be easily verified by induction that the number of edges of G_ℓ is $7 \cdot 2^\ell - 5$. So the ratio between greedy-weight and optimal for this instance is $\Omega(\frac{m}{\log m})$. ◀

While greedy-weight is most commonly used in applications, the approach was first presented as part of a general framework [25]: pick any optimality criteria for s - t paths that is *saturating* (i.e., fully decomposes at least one edge), and successively remove optimal paths. Since each path is saturating, the algorithm must decompose the flow in m or fewer paths. Another optimality criterion sometimes used in DNA assembly (e.g., in *vg-flow* [2]) is the *longest path* (with its maximum possible flow so that it is saturating). To adapt our construction of $(G_\ell, X_{\ell, 2^{\ell+1}})$ so that this approach has the same approximation ratio, consider $(G_\ell^*, X_{\ell, 2^{\ell+1}}^*)$, constructed as in $(G_\ell, X_{\ell, 2^{\ell+1}})$ except that we replace every backbone edge (u, v) with two edges, (u, w) and (w, v) . See [5, Figure 6] for an example. Then a path

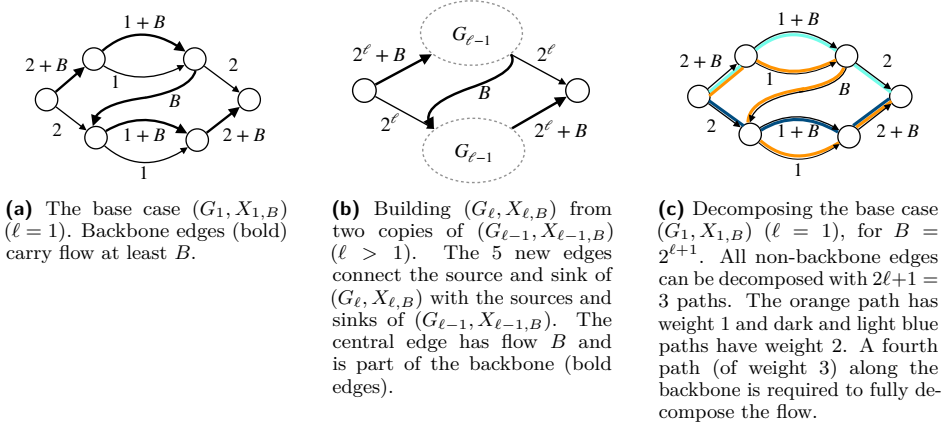


Figure 3 Construction for $(G_\ell, X_{\ell,B})$. Setting $B = 2^{\ell+1}$ gives MFD instances where greedy-weight uses $\Theta(\frac{m}{\log m})$ times more paths than optimal to decompose the flow.

along the backbone edges will be the longest from s to t and the above asymptotic analysis holds, since we no more than doubled the number of edges. Yet another optimality criterion, studied in [12] for its application to network routing, is the shortest path (again with its maximum possible flow). $(G_\ell^*, X_{\ell,2^{\ell+1}}^*)$ will also force this approach to take an exponential number of paths, since first the algorithm will take all $2^{\ell+1}$ weight-1 edges with $2^{\ell+1}$ different paths.

4.2 Greedy approximation for series-parallel graphs

As exploited in Section 4.1, one sticking point for greedy path removal algorithms is the fact that the width of a graph can increase after an edge is fully decomposed. We now show that if, in a particular instance, a graph does not increase its width during the execution of the algorithm, and greedy-weight can decompose “enough” flow at each step, then greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$.

If G is a directed graph and $X \geq 0$ a flow on G , we write $G|_X$ (G restricted to X) to mean the spanning subgraph of G made up of the edges $e \in E$ such that $X(e) \neq 0$. Conversely, if S is a subgraph of G , we write $X|_S$ (X restricted to S) to mean the pseudo-flow X only on the edges of S . In the case of $\text{MFD}_{\mathbb{N}}$, once an edge is fully decomposed, it cannot be used in future paths, possibly increasing the width of the graph that can be used to decompose the rest of the flow and sometimes triggering an increase of the size of a minimum flow decomposition as well. We call a graph *stable* if it does not have this issue.

► **Property 15 (Stable graph).** We say that G is *stable* if, for any non-negative flow X on G , it holds that $\text{width}(G|_X) \leq \text{width}(G)$.

Many useful $\text{MFD}_{\mathbb{N}}$ instances do in fact satisfy Property 15. For example, the first proof of MFD’s NP-hardness [25] was a reduction to a very simple graph of this form; this means that $\text{MFD}_{\mathbb{N}}$ restricted to stable graphs is also NP-hard.

The second property that we need is that there is always, during the execution of the algorithm, a path carrying “enough” flow from s to t .

► **Property 16 (Paths of large weight).** We say that G has *paths of large weight* if, for any flow $X \geq 0$ on G , there exists an s - t path in $G|_X$ carrying at least $|X|/\text{width}(G)$ flow.

Note that this property does not hold in general; see [5, Figure 7].

► **Lemma 17.** *Let $G = (V, E)$ be a graph, $\text{width}(G) \geq 2$, satisfying Properties 15 and 16. Greedy-weight uses at most $\lfloor \log |X| / \log \frac{\text{width}(G)}{\text{width}(G)-1} \rfloor + 1$ paths to decompose any flow $X : E \rightarrow \mathbb{N}$.*

Proof. Let $b = \text{width}(G)$. Since G satisfies Properties 15 and 16, greedy-weight removes a path of weight at least $|X'|/b$ at every step, where X' is the remaining flow of the corresponding step. As such, after c steps $|X'| \leq |X| \left(\frac{b-1}{b}\right)^c$. If $|X| \left(\frac{b-1}{b}\right)^c < 1$, then $|X'| = 0$, since $|X|$ and the weights of the removed paths belong to $\mathbb{N}$. Solving for c we obtain $c > \log |X| / \log \frac{b}{b-1}$. Therefore, greedy-weight takes (uses) at most $c = \lfloor \log |X| / \log \frac{b}{b-1} \rfloor + 1$ steps (paths). ◀

► **Theorem 18.** *Let $G = (V, E)$ be a graph satisfying Properties 15 and 16 and $X : E \rightarrow \mathbb{N}$ a flow. Greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ on (G, X) .*

Proof. Assume $X > 0$ (otherwise, replace G with $G|_X$). Thus, $b = \text{width}(G) \leq \text{mfd}_{\mathbb{N}}(G, X)$, since any flow-decomposition of X induces a path cover of E . If $b \leq 1$ greedy-weight finds an optimal solution. Otherwise $b \geq 2$, and Lemma 17 implies that greedy-weight is a $O\left(\frac{\log |X|}{b \log \frac{b}{b-1}}\right) = O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ ($b \log \frac{b}{b-1} = O(1)$ for $b \geq 2$). ◀

Finally, we define series-parallel graphs, and apply Theorem 18 to them, by proving (in [5, Appendix B]) that they satisfy Properties 15 and 16.

► **Definition 19** (Series-parallel graph [9]). *A graph is a two-terminal series-parallel (series-parallel for short) graph with terminal nodes s and t if:*

- *it consists of a single edge directed from s to t , and no other nodes, or*
- *it can be obtained from two (smaller) two-terminal series-parallel graphs G_1 and G_2 , with terminal nodes s_1, t_1 , and s_2, t_2 , respectively, by either*
 - *identifying $s = s_1 = s_2$ and $t = t_1 = t_2$ (parallel composition of G_1 and G_2), or*
 - *identifying $s = s_1$, $t_1 = s_2$, and $t = t_2$ (series composition of G_1 and G_2).*

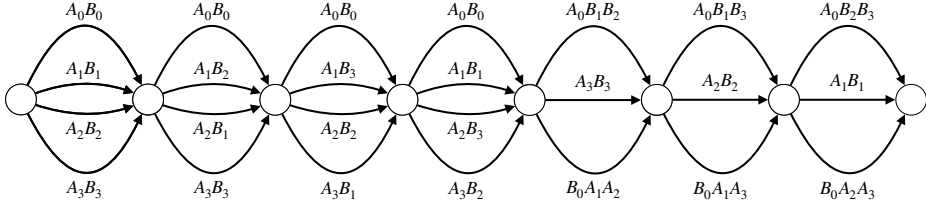
► **Corollary 20.** *Greedy-weight is a $O(\log |X|)$ -approximation for $\text{MFD}_{\mathbb{N}}$ on series-parallel graphs.*

5 Solving the k -Flow Weight Assignment Problem

In this section, we consider a restriction of MFD from [14] (see Figure 4 for an example).

► **Definition 21** (k -Flow Weight Assignment). *Given a flow $X : E \rightarrow \mathbb{Y}$ on a graph $G = (V, E)$ and a set of s - t paths $\{P_1, \dots, P_k\}$, the problem of finding an assignment of weights to the paths, such that they form a flow decomposition of (G, X) , is called k -Flow Weight Assignment (k -FWA). We write $k\text{-FWA}_{\mathbb{Y}}$ if we require the path weights to belong to $\mathbb{Y}$.*

Given k s - t paths, k -FWA can be solved by a linear system defined by $Lw = X$, where $X_j \in \mathbb{Y}$ is equal to the flow $X(e_j)$ of the edge e_j (we identify flows $X : E \rightarrow \mathbb{Y}$ with vectors $X \in \mathbb{Y}^m$) and L is the $m \times k$ 0/1 matrix with $L_{i,j} = 1$ if and only if path P_j crosses edge e_i . The resulting solution $w \in \mathbb{Y}^k$ is the weight assignment to each path. For a flow graph (G, X) , we denote by $L_{\mathbb{Y}} = L_{\mathbb{Y}}(P_1, \dots, P_k) = \{w \in \mathbb{Y}^k | X = \sum_{j=1}^k P_j w_j\}$ the linear system corresponding to the paths $P_1, \dots, P_k$.



■ **Figure 4** Paths A_i and B_i ($i \in \{0, 1, 2, 3\}$), each edge being labeled with the paths it appears in. Assign to each path A_i weight a_i , and to each path B_i weight b_i , such that $a_0 = b_0 = 3$, and $a_i = 6^{2i} + 1$ and $b_i = 6^{2i+1} + 5$ for $i = 1, 2, 3$. Define the flow X on G as $X = \sum_{i=0}^3 a_i A_i + \sum_{i=0}^3 b_i B_i$. Note that these weights are a solution of $k\text{-FWA}_{\mathbb{N}}$ on input (G, X) with given paths A_i, B_i ($i \in \{0, 1, 2, 3\}$).

We shortly discuss how to solve $k\text{-FWA}_{\mathbb{Z}}$. It is possible to find an integer-weighted solution or decide that it does not exist in polynomial time. The reason for this is that, having no non-negativity constraint, the linear system defined by the paths is a system of linear Diophantine equations. It is well known that integer solutions to such systems can be found in polynomial time; see, e.g., [19, Chapter 5].

On the other hand, solving $k\text{-FWA}_{\mathbb{N}}$ turns out to be more difficult. Here, the linear system contains the inequality $w \geq 0$. In fact, it was shown [14] that $k\text{-FWA}_{\mathbb{N}}$ is NP-hard. The program Toboggan [14] implements a linear FPT algorithm for $\text{MFD}_{\mathbb{N}}$. One step of the algorithm is to solve $k\text{-FWA}_{\mathbb{N}}$ using an ILP [14]. The authors state the following conjecture.

► **Conjecture 22** ([14]). *If $(P_1, \dots, P_k)$ are the paths of a minimum flow decomposition of (G, X) , then the linear system $L_{\mathbb{N}}(P_1, \dots, P_k)$ has full rank k .*

As mentioned in the introduction, if the conjecture turned out to be true, then Toboggan could avoid resorting to solving an ILP, since just solving the standard linear system at hand would return its unique solution. As observed by the authors, this would decrease the asymptotic worst case upper bound of Toboggan.

We show that this conjecture is false using a counterexample. Consider the input for $k\text{-FWA}_{\mathbb{N}}$ from Figure 4 and the solution therein. We now give another solution for $k\text{-FWA}_{\mathbb{N}}$ on this input, namely the following path weights: $a_0 = 5, b_0 = 1$, and $a_i = 6^{2i} + 2, b_i = 6^{2i+1} + 4$, for $i = 1, 2, 3$. One can easily verify that this is another solution to $k\text{-FWA}_{\mathbb{N}}$ on the input in Figure 4, thus proving that the rank of the corresponding linear system is strictly less than 8.

To disprove Conjecture 22, it remains to show that any flow decomposition contains at least 8 paths. Due to the technicality of this proof (and its exhaustive case-by-case analysis), in this paper we only explain the intuition behind the construction in Figure 4 and behind the correctness proof. However, as an additional check we also ran both Toboggan [14] and a recently developed ILP solver for $\text{MFD}_{\mathbb{N}}$ [7] on this instance, both returning $\text{mfd}_{\mathbb{N}}(G, X) = 8$.

The intuition is as follows. The graph can be divided into two parts: the graph induced by the first 5 vertices in topological order (left part) and the one induced by the last 5 vertices (right part). The exponential growth of the paths A_i and B_i for growing i , together with the different permutations of the paired labels $A_i B_j$ on the left part, fix the choice of the paths A_i and B_i for $i = 1, 2, 3$. This allows us to interpret flow decompositions of less than 8 paths as decompositions with 8 paths, where either A_0 or B_0 carries weight 0. Consider a flow decomposition where we assign two paths of weights λ_1 and λ_2 on the edges labeled $A_0 B_0$. For any $\delta \geq 0$, $(\lambda_1 - \delta) + (\lambda_2 + \delta) = a_0 + b_0$ and equivalently for all other edges on the left part. If we decrease λ_1 by some $\delta > 0$, the weights of B_1 and B_2 each increase by $\delta/2$. And thus, δ must be even. Due to the parity of a_0 and b_0 , they can never reach 0.

6 Conclusions

In this paper we have shown for the first time that width, a natural lower bound for MFD, is also useful when investigating its approximability. On the one hand, using width is a key insight in understanding where greedy path removal heuristics fail. On the other hand, graphs where width is well-behaved (e.g., series-parallel graphs) have a guaranteed approximation factor. Moreover, when combined with parity arguments, i.e., about parity fixing unitary flows, and a width-sized decomposition of such flows, it guarantees an even better approximation factor for $\text{MFD}_{\mathbb{Z}}$ for all DAGs. Finally, we have corroborated the complexity gap between the positive integer and the full integer case by disproving a conjecture from [14] (also motivating the heuristic in [20]), which would have had sped up their FPT algorithm for $\text{MFD}_{\mathbb{N}}$.

Our results open up new avenues for further research on MFD. For example, can the width help find larger classes of graphs for which some greedy path removal (or even some sort of greedy *path cover* removal) algorithms have a guaranteed approximation factor? Can we get $\Omega(n)$ worst case approximation ratio of greedy-weight for dense graphs without parallel edges? Can the power-of-two decomposition approach be applied with other factors besides two? Can better path cover-like lower bounds help (e.g., path covers which cannot use an edge more times than its flow value, also computable in polynomial time)? How do our algorithms perform in practice?

References

- 1 Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: Theory, algorithms and applications. *New Jersey: Prentice-Hall*, 1993.
- 2 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.
- 3 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.
- 4 Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. Robust and adaptive network flows. *Operations Research*, 61(5):1218–1242, 2013. doi:10.1287/opre.2013.1200.
- 5 Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams. Width helps and hinders splitting flows. *CoRR*, 2022. doi:10.48550/ARXIV.2207.02136.
- 6 Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *SODA 2022 - ACM-SIAM Symposium on Discrete Algorithms*, pages 359–376, 2022. doi:10.1137/1.9781611977073.18.
- 7 Fernando H. C. Dias, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. Fast, Flexible, and Exact Minimum Flow Decompositions via ILP. *arXiv*, arXiv:2201.10923, 2022. To appear in the Proceedings of RECOMB 2022 – 26th Annual International Conference on Research in Computational Molecular Biology. arXiv:2201.10923.
- 8 Robert P Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: <http://www.jstor.org/stable/1969503>.
- 9 David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992. doi:10.1016/0890-5401(92)90041-D.
- 10 Delbert R Fulkerson. Note on Dilworth’s decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4):701–702, 1956.
- 11 Thomas Gatter and Peter F Stadler. Ryūto: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20(1):1–14, 2019.

- 12 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.
- 13 A. Jain and N. Chandrasekharan. An efficient parallel algorithm for min-cost flow on directed series-parallel networks. In *Proceedings Seventh International Parallel Processing Symposium*, pages 188–192, 1993. doi:10.1109/IPPS.1993.262879.
- 14 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.
- 15 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.
- 16 Nils Olsen, Natalia Kliwer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time-space network models. *Central European Journal of Operations Research*, 2020. doi:10.1007/s10100-020-00705-6.
- 17 James B. Orlin. Max flows in $O(nm)$ time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.
- 18 Mihaela Perteau, Geo M Perteau, 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.
- 19 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., 1986.
- 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 Mingxiang Teng, Michael I Love, Carrie A Davis, Sarah Djebali, Alexander Dobin, Brenton R Graveley, Sheng Li, Christopher E Mason, Sara Olson, Dmitri Pervouchine, et al. A benchmark for rna-seq quantification pipelines. *Genome biology*, 17(1):1–12, 2016.
- 22 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.
- 23 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.
- 24 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982. doi:10.1137/0211023.
- 25 Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe 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.
- 26 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.

Paper II

II

Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams

Width Helps and Hinders Splitting Flows

In *ACM Transaction on Algorithms*,
20(2):13, 2024.

Copyright © Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams.



Width Helps and Hinders Splitting Flows

MANUEL CÁCERES, MASSIMO CAIRO, and ANDREAS GRIGORJEW, Department of Computer Science, University of Helsinki, Helsinki, Finland

SHAHBAZ KHAN, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee, India

BRENDAN MUMEY, School of Computing, Montana State University, Bozeman, United States

ROMEO RIZZI, Department of Computer Science, University of Verona, Verona, Italy

ALEXANDRU I. TOMESCU, Department of Computer Science, University of Helsinki, Helsinki, Finland

LUCIA WILLIAMS, Department of Computer Science, University of Montana, Missoula, United States

Minimum flow decomposition (MFD) is the NP-hard problem of finding a smallest decomposition of a network flow/circulation X on a directed graph G into weighted source-to-sink paths whose weighted sum equals X . We show that, for acyclic graphs, considering the *width* of the graph (the minimum number of paths needed to cover all of its edges) yields advances in our understanding of its approximability. For the version of the problem that uses only non-negative weights, we identify and characterise a new class of *width-stable* graphs, for which a popular heuristic is a $O(\log \text{Val}(X))$ -approximation ($\text{Val}(X)$ being the total flow of X), and strengthen its worst-case approximation ratio from $\Omega(\sqrt{m})$ to $\Omega(m/\log m)$ for sparse graphs, where m is the number of edges in the graph. We also study a new problem on graphs with cycles, Minimum Cost Circulation Decomposition (MCCD), and show that it generalises MFD through a simple reduction. For the version allowing also negative weights, we give a $(\lceil \log \|X\| \rceil + 1)$ -approximation ($\|X\|$ being the maximum absolute value of X on any edge) using a power-of-two approach, combined with parity fixing arguments and a decomposition of unitary circulations ($\|X\| \leq 1$), using a generalised notion of width for this problem. Finally, we disprove a conjecture about the linear independence of minimum (non-negative) flow decompositions posed by Kloster et al. [2018], but show that its useful implication (polynomial-time assignments of weights to a given set of paths to decompose a flow) holds for the negative version.

CCS Concepts: • **Theory of computation** → **Approximation algorithms analysis; Network flows;**

Additional Key Words and Phrases: Flow decomposition, graph width

A preliminary version of this paper appeared in the 30th Annual European Symposium on Algorithms, ESA 2022, September 5–9, 2022, Berlin/Potsdam, Germany.

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, SAFE BIO), partially by the Academy of Finland (grants No. 322595, 328877, 352821, 346968), and partially by the National Science Foundation (NSF) (grants No. 1661530, 1759522). Authors’ addresses: M. Cáceres, M. Cairo, A. Grigorjew, and A. I. Tomescu, Department of Computer Science, University of Helsinki, Helsinki 00014, Finland; e-mails: manuel.caceresreyes@helsinki.fi, cairomassimo@gmail.com, andreas.grigorjew@helsinki.fi, alexandru.tomescu@helsinki.fi; S. Khan, Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, Roorkee 247667, India; e-mail: shahbaz.khan@cs.iitr.ac.in; B. Mumey, School of Computing, Montana State University, Bozeman 59717, USA; e-mail: brendan.mumey@montana.edu; R. Rizzi, Department of Computer Science, University of Verona, Verona 37129, Italy; e-mail: romeo.rizzi@univr.it; L. Williams, Department of Computer Science, University of Montana, Missoula 59812, USA; e-mail: lgw2@uw.edu.



This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).

ACM 1549-6325/2024/03-ART13

<https://doi.org/10.1145/3641820>

ACM Reference Format:

Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I. Tomescu, and Lucia Williams. 2024. Width Helps and Hinders Splitting Flows. *ACM Trans. Algor.* 20, 2, Article 13 (March 2024), 20 pages. <https://doi.org/10.1145/3641820>

1 INTRODUCTION

Minimum flow decomposition (MFD) is the problem of finding a smallest sized decomposition of a network flow X on directed graph $G = (V, E)$ into weighted source-to-sink paths whose weighted sum equals X . We focus on the case where path weights are restricted to be integers ($\text{MFD}_{\mathbb{Z}}$) or natural numbers ($\text{MFD}_{\mathbb{N}}$). It is a textbook result [Ahujia et al. 1993] that if G is acyclic (a DAG) a decomposition using no more than $m = |E|$ paths always exists. However, MFD is strongly NP-hard [Vatinlen et al. 2008], even on DAGs, and even when the flow values come only from $\{1, 2, 4\}$ [Hartman et al. 2012]. Recent work has shown that the problem is FPT in the size of the minimum decomposition [Kloster et al. 2018] and that it can be formulated as an ILP of quadratic size [Dias et al. 2022].

While difficult to solve, MFD is a key step in many applications. For example, MFD on DAGs is used to reconstruct biological sequences such as RNA transcripts [Bernard et al. 2014; Dias et al. 2023; Gatter and Stadler 2019; Pertea et al. 2015; Tomescu et al. 2015, 2013; Williams et al. 2019] and viral strains [Baaijens et al. 2020]. MFD can also be used to model problems in networking [Hartman et al. 2012; Mumey et al. 2015; Vatinlen et al. 2008] and transportation planning [Olsen et al. 2020], although in some of these applications there may be cycles in the input. Despite the ubiquity of the MFD problem, the gap in our knowledge about the approximability of MFD is large. It is known [Hartman et al. 2012] that MFD (even on DAGs) is APX-hard (i.e., there is some $\epsilon > 0$ such that it is NP-hard to approximate within a $(1 + \epsilon)$ factor), so in particular, MFD does not admit a PTAS, unless $P=NP$. Furthermore, the best known approximation ratio is $\lambda^{\log \|X\|} \log \|X\|$ [Mumey et al. 2015], where λ is the length of the longest source-to-sink path and $\|X\|$ is the largest flow value in the network. In this work, we attempt to fill in some of the gaps between these results.

A natural lower bound for the size of an MFD of a DAG is the size of a *minimum path cover* of the set of edges with non-zero flow (i.e., the minimum number of paths such that every such edge appears in *at least* one path)—this size is called the *width* of the network. This trivially holds because every flow decomposition is also such a path cover. These two notions are analogies of the more standard notions of path cover and width of the *node set*. The node-variants are classical concepts, with algorithmic results dating back to Dilworth [1950] and Fulkerson [1956]. Despite this, the width has not been given any attention in the MFD problem, and in particular it has never been used in approximation algorithms to our knowledge. In this article, we show that the width can play a key role both in the analysis of popular heuristics, and in obtaining the first approximation algorithm for a natural generalisation of MFD.

We start by considering the connections between the width and a popular heuristic algorithm for $\text{MFD}_{\mathbb{N}}$ which we call *greedy-weight*¹ [Vatinlen et al. 2008], which builds a flow decomposition by successively choosing the path that can carry the largest flow. Greedy-weight is commonly used in applications (see e.g., Baaijens et al. [2020], Pertea et al. [2015], and Tomescu et al. [2013] among many), and it seems to be mentioned in nearly every publication addressing flow decomposition. First, on sparse graphs we improve (i.e., increase) the worst-case lower bound for

¹Previous work has consistently referred to this algorithm as *greedy-width*. To avoid confusion with the width of the graph, we introduce the name *greedy-weight* in this work.

the greedy-weight approximation factor from $\Omega(\sqrt{m})$ [Hartman et al. 2012], showing for the first time that greedy-weight can be exponentially worse than the optimum:

THEOREM 1. *The approximation ratio for greedy-weight on $\text{MFD}_{\mathbb{N}}$ is $\Omega(m/\log m)$ for sparse graphs, in the worst case.*

For this we use a class of sparse graphs where the optimum flow decomposition has size $O(\log m)$ whereas the greedy-weight algorithm returns a solution of size $\Omega(m)$, only a constant factor away from the trivial decomposition. The key to this new bound is to design an input where the width increases exponentially when a path is greedily removed. We also show that the same bound also holds for other greedy heuristics choosing instead the longest or shortest paths. Second, we identify a new class of graphs, defined by the property that their width does not increase as source-to-sink paths are removed (see Definition 11 of width-stable graphs). We show a relation of width-stable graphs to funnels: precisely, a graph is not width-stable iff it contains a funnel subgraph and a certain *central* path. This is precisely the structure of the class of sparse graphs improving the approximation ratio of greedy-weight in Theorem 1. We also show that width-stability enables greedy-weight to remove paths of large enough flow (Lemma 13), leading to the following result, with $\text{Val}(X)$ being equal to the total flow of the graph:

THEOREM 2. *Let $G = (V, E)$ be a width-stable graph and $X : E \rightarrow \mathbb{N}$ a flow. Greedy-weight is a $O(\log \text{Val}(X))$ -approximation for $\text{MFD}_{\mathbb{N}}$ on (G, X) .*

A notable example of width-stable graphs is the class of *series-parallel graphs*; see [Eppstein 1992; Valdes et al. 1982] for fast recognition algorithms and pointers to other NP-hard problems that are easier on this class of graphs. Series-parallel graphs are also of great interest for network flow problems (see, e.g., Bertsimas et al. [2013] and Jain and Chandrasekharan [1993]). Theorems 1 and 2 show that greedy-weight's approximation ratio is highly linked to the width stability of the graph.

In Section 4, we continue with a generalised version of MFD, **Minimum Cost Circulation Decomposition (MCCD)**, on directed graphs with cycles and no sinks or sources, and a cost function on the edges. Instead of decomposing a flow into weighted paths, we decompose a circulation into *weighted circulations* and minimise the total cost of the circulations, and instead of the width, a natural lower bound for this problem is the **minimum cost of a circulation cover (mccc)**. Decomposing into circulations rather than paths is a natural generalisation, as paths can be considered as value 1 flows themselves. Additionally, we also consider a relaxation in which the flow/circulation decomposition might use negative integer weights on flows/circulations, rather than strictly positive weights as has traditionally been considered [Hartman et al. 2012; Kloster et al. 2018; Vatinlen et al. 2008]. An important observation that we leverage for this variant (unlike the positive-only version) is that the width/mccc stays constant as flow is chosen and removed. Using this, we give a $(\lceil \log \|X\| \rceil + 1)$ -approximation algorithm for this variant.

We denote the problem versions for non-negative path weights and integer path weights by $\text{MCCD}_{\mathbb{N}}$ and $\text{MCCD}_{\mathbb{Z}}$ as well as $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$, respectively, throughout the article. While $\text{MCCD}_{\mathbb{Z}}$ and $\text{MFD}_{\mathbb{Z}}$ are natural versions of the problem, they have not been previously considered in the MFD literature to our knowledge. However, $\text{MFD}_{\mathbb{Z}}$ can also have natural applications, since by applying $\text{MFD}_{\mathbb{Z}}$ on the difference between two flows, one can minimally explain the *differences* between them, for example, to explain the differences in RNA expression between two tissue samples with the fewest number of up/down regulated transcripts, which is often the goal of RNA sequencing experiments [Teng et al. 2016]. Our approximation follows a *power-of-two* approach, where the weights of the flows/circulations chosen are (positive or negative) powers of two. More specifically, observe that if all circulation values are even, then one can divide them by

2 and obtain a circulation X with smaller $\|X\|$ whose decomposition can be transformed back into a decomposition of X . In order to obtain such an even circulation, we prove a basic property that can be of independent interest: given any integer circulation X , there exists a *unitary* circulation (its values are 0, +1, or -1) Y , such that $X + Y$ is even on every edge (Lemma 21). In addition, given a unitary circulation Y , we show that Y can be decomposed into circulations of total cost no more than mccc (Lemma 23). We obtain the $(\lceil \log \|X\| \rceil + 1)$ -approximation ratio (Theorem 3) by iteratively removing the unitary circulation, dividing all circulation values by 2, and preprocessing the graph so that the mccc is a lower bound on the size of the $\text{MCCD}_{\mathbb{Z}}$. Summarised, we show:

THEOREM 3. *$\text{MCCD}_{\mathbb{Z}}$ can be approximated with a factor of $\log \lceil \|X\| \rceil + 1$ in runtime $O(n \log m(m + n \log n) + m \log \|X\|)$.*

By Corollary 26, we additionally obtain the result for $\text{MFD}_{\mathbb{Z}}$. Notably, the runtime of the algorithm does not depend on the cost function.

Finally, in Section 5, we consider a closely related problem, called *k-Flow Weight Assignment* [Kloster et al. 2018]. In addition to the flow X , in this problem we are also given a set of k paths, and we need to decide if there is an assignment of weights to the paths such that they form a decomposition of X . If the weights belong to $\mathbb{N}$, this was shown to be NP-complete in Kloster et al. [2018]. In this work, we first observe that in the same way that allowing negative integer weights simplifies the approximability of MFD, allowing weights to belong to $\mathbb{Z}$ fully changes the complexity of the *k-Flow Weight Assignment Problem*, making it polynomial. This is due to the fact that the linear system defined by the given paths loses its only inequality of restricting the weights to positive integers. It thus transforms an ILP to a system of linear diophantine equations, which can be solved in polynomial time (see e.g. Schrijver [1986]). Second, we consider a conjecture from Kloster et al. [2018] stating that if the weights belong to $\mathbb{N}$, and k is the size of a $\text{MFD}_{\mathbb{N}}$ for X , then the problem admits a unique solution (i.e., a unique assignment of weights to the given paths). If true, this would speed up the FPT algorithm of Kloster et al. [2018] for $\text{MFD}_{\mathbb{N}}$, because a step solving an ILP could be executed by solving a standard linear program returning a rational solution and checking if the (supposedly unique) solution to this system is integer. Moreover, the same conjecture (with the same implication) was also a motivation behind the greedy algorithm of Shao and Kingsford [2017] for $\text{MFD}_{\mathbb{N}}$. In this article, we disprove the conjecture of Kloster et al. [2018], further corroborating the gap between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$.

2 PRELIMINARIES

In Sections 3 and 5, we are given a directed acyclic graph $G = (V, E)$. Without loss of generality, we assume a unique source s and a unique sink t with no in-edges and no out-edges, respectively; otherwise, the graph can be converted to such a graph by adding a pseudo source and sink and connecting them to all sources and sinks, respectively. We denote by $\deg^+(v)$ and $\deg^-(v)$ the out- and indegree of a vertex v , respectively. While MFD are also studied for graphs with cycles (see, e.g., Hartman et al. [2012] and Vatinlen et al. [2008]), the task is still to decompose into simple paths, and so our inapproximability result on DAGs in Section 3 also applies for graphs with cycles. In Section 4, we consider directed graphs $G = (V, E, c)$ with no sources or sinks, where $c : E \rightarrow \mathbb{R}_{\geq 0}$ is a cost function. Such graphs cannot be acyclic. We use n and m to denote the number of nodes and edges of G , respectively. For both kinds of graphs, we call functions $X : E \rightarrow \mathbb{Y}$ *pseudo-flows*,² where $\mathbb{Y} \in \{\mathbb{N}, \mathbb{Z}\}$ is some set of allowed flow values. We treat pseudo-flows as vectors over E and use the notation $X + Y$ and aX to denote the (element-wise) sum of pseudo-flows and

²Commonly in the literature, (pseudo-)flows are additionally required to be skew-symmetric and to be upper-bounded by some capacity function on the edges.

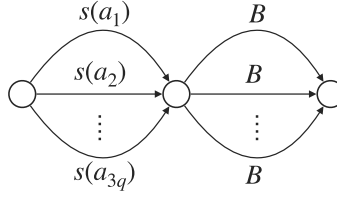


Fig. 1. The reduction of 3-Partition to MFD from Vatinlen et al. [2008]. The 3-Partition instance consists of a set $A = \{a_1, \dots, a_{3q}\}$, where every a_i has a positive integer size $s(a_i)$, and a positive integer B , such that $B/4 < s(a_i) < B/2$ holds for every $a_i \in A$. The question is whether A can be partitioned into q disjoint subsets, each of 3 elements and of size B . The MFD series-parallel (see Definition 15) reduction consists of a subgraph obtained by the *parallel composition* of $3q$ edges with flow values $s(a_1), \dots, s(a_{3q})$, and a subgraph obtained by the *parallel composition* of q edges, each with flow value B . These two graphs are composed with the *series composition*. Intuitively, because $B/4 < s(a_i) < B/2$ holds for every a_i , the MFD consists of exactly $3q$ paths of weights $s(a_1), \dots, s(a_{3q})$, and each edge on the right-hand subgraph is traversed by exactly three paths whose weights sum to B , giving thus the partition of A . Moreover, since the first $3q$ edges need to be decomposed, the previous decomposition is minimum even if negative weights are allowed, making $\text{MFD}_{\mathbb{Z}}$ NP-hard.

multiplication by a scalar, respectively. The numbers 0 and 1 also denote (depending on context) pseudo-flows that are 0 (resp. 1) on every edge. We write $X \leq Y$ (and similarly $<$) to mean $X(u, v) \leq Y(u, v)$ for every $(u, v) \in E$.

Given a DAG G , a *flow* is a pseudo-flow satisfying conservation of flow (incoming flow equal to outgoing flow) on internal nodes $V \setminus \{s, t\}$. A pseudo-flow satisfying the conservation of flow on all nodes is called a *circulation*. We sometimes refer to the value $X(e)$ for a flow/circulation X as the flow of the edge e . It is known that the sum of two flows/circulations $X + Y$, the multiplication of a flow/circulation with by a scalar aX , and the empty pseudo-flow 0 are themselves flows/circulations. For a flow X , let $\text{Val}(X)$ denote the total flow out of s (or equivalently into t by flow conservation). Note that $\text{Val}(X)$ can be negative. Given an s - t path P , denote by $P[u..v]$ the subpath of P going from $u \in V$ to $v \in V$ and let P also denote the flow defined by setting *one* to every edge in P and 0 to every other edge. With these definitions, we are ready to formally define MFD.

Definition 4. Given a flow X , a *flow decomposition* of (G, X) of size k is a family of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ with weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $X = w_1 P_1 + \dots + w_k P_k$.

Definition 5. For a flow X , let $\text{mfd}_{\mathbb{Y}}(G, X)$ be the smallest size of a flow decomposition of (G, X) with weights in $\mathbb{Y}$.

We omit $\mathbb{Y}$ if it is clear from the context. We call the problem of producing a flow decomposition of (G, X) of minimum size the *MFD problem*.

In Section 3, we study $\text{MFD}_{\mathbb{N}}$ ($0 \in \mathbb{N}$), and in Section 4, we study $\text{MFD}_{\mathbb{Z}}$ and its generalisation $\text{MCCD}_{\mathbb{Z}}$. Note that the reduction showing $\text{MFD}_{\mathbb{N}}$ to be strongly NP-hard from Vatinlen et al. [2008] also holds for $\text{MFD}_{\mathbb{Z}}$ (see Figure 1). However, a flow with non-negative values may admit a decomposition using fewer paths if negative weights are allowed, as shown in Figure 2. We explore further differences between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ in Sections 4 and 5.

Let $\|X\| = \max_{(u,v) \in E} |X(u, v)|$ denote the maximum norm on flows or circulations. In particular, notice that if $\mathbb{Y} \subseteq \mathbb{Z}$, then $\|X\| \leq 1$ means that $X(u, v) \in \{0, \pm 1\}$ for every $(u, v) \in E$. Let $X \equiv_2 Y$ iff X and Y have the same parity everywhere, that is, for every $(u, v) \in E$, we have that $X(u, v)$ is odd iff $Y(u, v)$ is odd.

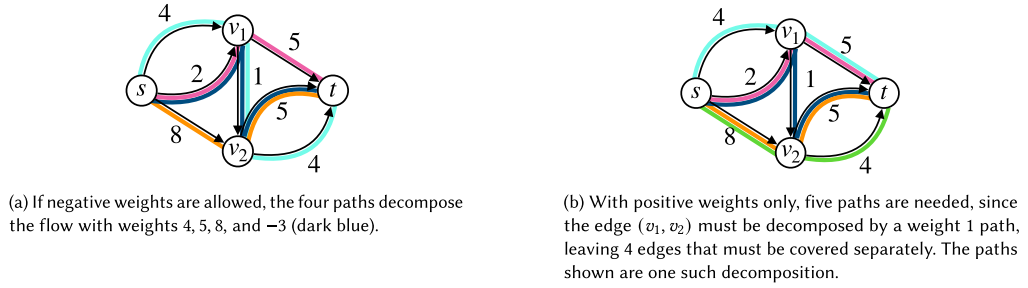


Fig. 2. A positive flow admitting a decomposition into four paths only if negative weights are allowed.

Definition 6. Given $S \subseteq E$, we define $\text{width}_S(G)$ as the minimum number of s - t paths in a DAG G needed to cover all edges of S . If $S = E$ we just write $\text{width}(G)$.

The width is the main combinatorial tool that we use for our approximation results, and we will show in Section 3, that it is highly linked to the approximation performance of greedy-weight. Just like its more common node variant, $\text{width}(G)$ can be computed in $O(mn)$ time. As described by, for example, Ahuja et al. [1993] and Cáceres et al. [2022], this is done by reduction to a min-flow instance with demand one on every edge; the minimum flow of this instance is $\text{width}(G)$, and the flow can be found by reduction to a max-flow instance. Moreover, the problem can be relaxed to only require the coverage of $S \subseteq E$ and solved in the same running time by setting the demands only on the edges of S .

LEMMA 7 (AHUJIA ET AL. [1993] AND ORLIN [2013]). Let $G = (V, E)$ be a DAG, and $S \subseteq E$. A flow $C : E \rightarrow \mathbb{N}$ can be computed in $O(mn)$ time, such that $C(e) \geq 1$ for all $e \in S$ and $|C| = \text{width}_S(G)$.

The flow C with total flow $\text{width}_S(G)$ suffices and we do not need to calculate a path cover achieving that minimum. However, we note that it can be directly computed given the flow C . We can think of this path cover as a flow decomposition of C into $\text{width}_S(G)$ weight-one paths, which can be found by greedily removing such paths from C until it is completely decomposed. Since each path has no more than $n - 1$ edges and since $\text{width}_S(G) \leq m$, the overall runtime of finding the path cover is $O(mn)$. Similarly, every path cover $P_1, \dots, P_\ell$ of G defines a flow C on G : $C = P_1 + \dots + P_\ell$, and we say that C is the induced flow of the path cover $(P_i)_1^\ell$.

Definition 8. In a directed graph $G = (V, E)$ we call a subset $C \subseteq E$ an *antichain* of G if all edges in C are pairwise parallel, that is there exists for no pair of edges in C a path in G leading from one edge to the other.

It can be shown with straight forward arguments that $\text{width}(G) = |C|$ for a maximum (sized) antichain C of G .

3 WIDTH MATTERS FOR GREEDY APPROACHES

Since the difference of two flows is still a flow, it is very natural to consider successively removing the simplest type of flow – that is to say, paths – as an algorithmic strategy for $\text{MFD}_{\mathbb{N}}$. Indeed, the particular greedy path removal strategy of finding a *heaviest path* (*greedy-weight*) is commonly used as a heuristic in applications (e.g., Baaijens et al. [2020], Hartman et al. [2012], Pertea et al. [2015], and Tomescu et al. [2013]) and it seems to be mentioned in nearly every article addressing flow decomposition. More formally, a path P is said to *carry* flow p if $X(u, v) \geq p$ for all edges (u, v) of P (in particular, a v - v path carries infinite flow). A *heaviest path* is an s - t path carrying the largest flow. Such a path can be easily found in linear time in the size of the DAG by dynamic programming (see, e.g., Vatinlen et al. [2008]).

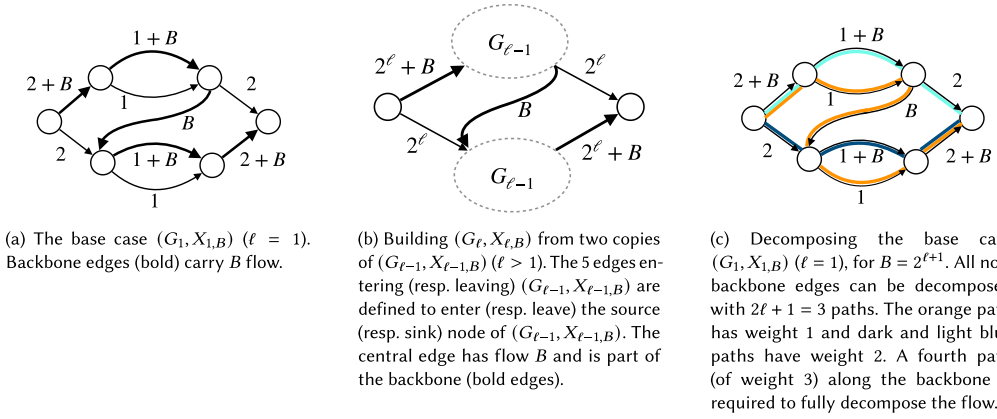


Fig. 3. Construction for $(G_\ell, X_{\ell,B})$. Setting $B = 2^{\ell+1}$ yields $\text{MFD}_{\mathbb{N}}$ instances where greedy-weight uses $\Theta(\frac{m}{\log m})$ times more paths than optimal to decompose the flow.

3.1 Width Hinders Greedy on $\text{MFD}_{\mathbb{N}}$

We define a family of $\text{MFD}_{\mathbb{N}}$ instances $(G_\ell, X_{\ell,B})$, depending on two parameters $\ell \in \mathbb{N} \setminus \{0\}$ and $B \in \mathbb{N}$. The family is defined recursively on ℓ . The base case $(G_1, X_{1,B})$ for $\ell = 1$ is shown in Figure 3(a). For $\ell > 1$, we build $(G_\ell, X_{\ell,B})$ from two disjoint copies of $(G_{\ell-1}, X_{\ell-1,B})$, by adding five extra edges and flow values as shown in Figure 3(b). We call the edge connecting the two copies of $G_{\ell-1}$ a *central edge*. Edges whose flow value depends on B are called *backbone edges*, and they form a s - t path. By choosing $B = 2^{\ell+1}$, we show that the flow $X_{\ell, 2^{\ell+1}}$ can be decomposed using a number of paths linear in ℓ , thanks to the heavy backbone edges, whereas the greedy-weight algorithm fully saturates the central edges with its first path and is left with a remaining flow requiring $2^{\ell+1}$ paths to be decomposed.

LEMMA 9. *For the graph G_ℓ with flow $X_{\ell, 2^{\ell+1}}$, greedy-weight uses $1 + 2^{\ell+1}$ paths to decompose $X_{\ell, 2^{\ell+1}}$.*

PROOF. We first show that the heaviest path in $X_{\ell, 2^{\ell+1}}$ follows every backbone edge from s to t . Certainly this path carries $2^{\ell+1}$ flow, and no more, since every backbone edge has flow at least $2^{\ell+1}$ and every central edge has flow exactly $2^{\ell+1}$. To see that there is no heavier path, observe that all the non-backbone edges have flow value strictly less than $2^{\ell+1}$ by construction. After removing that path with weight $2^{\ell+1}$, all the central edges are completely decomposed. The remaining graph and flow, without central edges, has $2^{\ell+1}$ weight-1 edges that are pairwise non-reachable using only non-zero flow edges (four edges for each of the $2^{\ell-1}$ copies of G_1), each of which must be covered by a different path of weight 1 (and these paths fully decompose the flow). $\square$

LEMMA 10. *For the graph G_ℓ , flow $X_{\ell, 2^{\ell+1}}$ can be decomposed using $2\ell + 2$ paths.*

PROOF. Using induction, we first prove that we can use $2\ell + 1$ paths to decompose all of the flow of $X_{\ell, 2^{\ell+1}}$ on the non-backbone edges of G_ℓ , and that these paths have total weight $2^{\ell+2} - 3$. When $\ell = 1$ (see Figure 3(c)), we can decompose both the flow-1 edges with a single path of weight 1, which goes through the central edge. Moreover, we can decompose the flow-2 edges with two paths of weight 2, without using the central edge. These paths have $5 = 2^{1+2} - 3$ total flow. We now assume that the claims hold for $\ell = k$ and prove it for $\ell = k + 1$. Consider the graph G_{k+1} . By assumption, the non-backbone edges in every copy of G_k are fully decomposed by $2k + 1$ paths, and the total flow of those paths is $2^{k+2} - 3$. Note that these paths are s - t paths in G_k , but they must

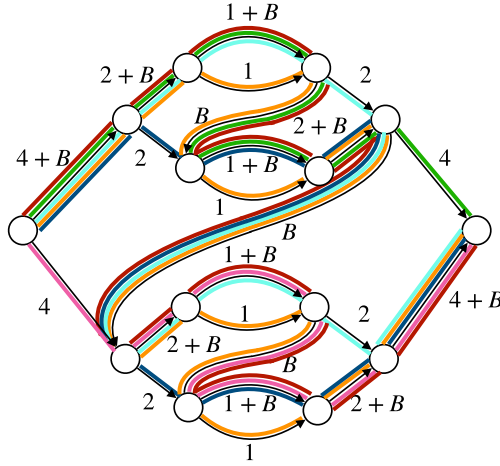


Fig. 4. $(G_2, X_{2, B=2^{\ell+1}})$ with a $2\ell + 2 = 6$ flow decomposition. Orange (weight 1) and dark and light blue (weight 2) decompose the non-backbone edges in the two copies of G_1 . Green and pink (weight 4) decompose the non-backbone edges added in G_2 . Dark red (weight 3) is the additional path needed to fully decompose the flow.

be extended to be s - t paths in G_{k+1} . Because there is a central edge with weight $2^{(k+1)+1} = 2^{k+2} > 2^{k+2} - 3$ from one copy of G_k to the other, it is possible to route the $2k + 1$ paths from the first G_k to the second using the central edge. Additionally, the backbone edges from s to the first copy of G_k and from the second copy of G_k to t have flow $2^{k+1} + 2^{k+2}$, so they can be used to complete the routing of the paths from s to t . Then we can use two additional paths of weight 2^{k+1} each to decompose the flow of the two new non-backbone edges; one using the backbone path of the upper G_k (extended by the upper edges of G_{k+1}), and analogously, the other through the backbone path of the lower G_k , which is possible since the paths obtained inductively only decompose 2^{k+1} of the backbone flow of G_k whereas now 2^{k+2} backbone flow must be decomposed. As such, we use $2k + 1 + 2 = 2(k + 1) + 1$ paths with total flow $2^{k+2} - 3 + 2(2^{k+1}) = 2^{(k+1)+2} - 3$, as required. By the previous, given any graph G_ℓ , we can decompose all non-backbone edges using $2\ell + 1$ paths. Note that removing these weighted paths from $X_{\ell, 2^{\ell+1}}$ yields a flow on G_ℓ (of value 3). Because the remaining edges (all backbone) form a path from s to t , and the remaining edge values form a flow on G_ℓ , all remaining edges must have the same flow value, and can be covered by one path. Thus, $2\ell + 2$ paths are sufficient to decompose $X_{\ell, 2^{\ell+1}}$. See Figure 4 for an example when $\ell = 2$. $\square$

THEOREM 1. *The approximation ratio for greedy-weight on $MFD_{\mathbb{N}}$ is $\Omega(m/\log m)$ for sparse graphs, in the worst case.*

PROOF. By Lemmas 9 and 10, greedy-weight uses $\Theta(2^\ell)$ paths to decompose the flow $X_{\ell, 2^{\ell+1}}$ described above, whereas it is possible to decompose the flow with only $\Theta(\ell)$ paths. It can be easily verified by induction that the number of edges of G_ℓ is $7 \cdot 2^\ell - 5$. So the ratio between greedy-weight and the optimal for this instance is $\Omega(\frac{m}{\log m})$. $\square$

While greedy-weight is most commonly used in applications, the approach was first presented as part of a general framework [Vatinlen et al. 2008]: pick any optimality criteria for s - t paths that is *saturating* (i.e., fully decomposes at least one edge), and successively remove optimal paths. Since each path is saturating, the algorithm must decompose the flow in m or fewer paths. Another optimality criterion sometimes used in DNA assembly (e.g., in *vg-flow* [Baaijens et al. 2020]) is the *longest path* (with its maximum possible flow so that it is saturating). To adapt our construction

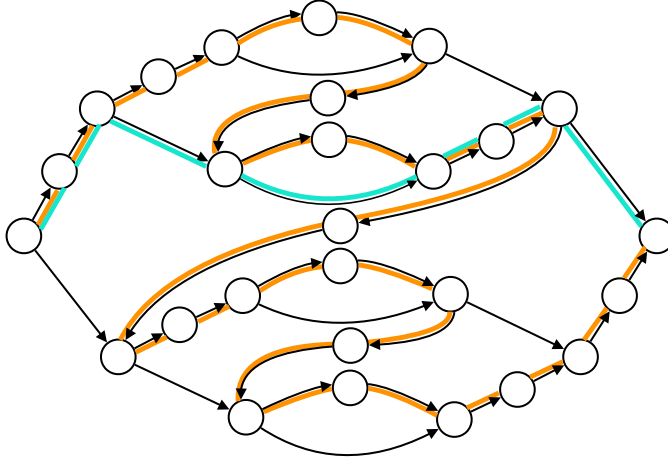


Fig. 5. By subdividing the backbone edges, choosing longest paths (orange) and shortest paths (blue) both give the same approximation ratio as choosing heaviest paths in the original flow $X_{\ell, 2^{\ell+1}}$.

of $(G_\ell, X_{\ell, 2^{\ell+1}})$ so that this approach yields the same approximation ratio, consider $(G_\ell^*, X_{\ell, 2^{\ell+1}}^*)$, constructed as in $(G_\ell, X_{\ell, 2^{\ell+1}})$ except that we split every backbone edge (u, v) into two edges, (u, w) and (w, v) . See Figure 5 for an example. Then the path along the backbone edges will be the longest from s to t and the previous asymptotic analysis still holds, since we no more than doubled the number of edges (and the number of edges of the new construction is still $\Theta(2^\ell)$). Yet another optimality criterion, studied in Hartman et al. [2012] for its application to network routing, is the shortest path (again with its maximum possible flow). $(G_\ell^*, X_{\ell, 2^{\ell+1}}^*)$ will also force this approach to take an exponential number of paths, since first the algorithm will decompose all $2^{\ell+1}$ weight-1 edges with $2^{\ell+1}$ different paths.

3.2 Greedy Approximation for Width-Stable Graphs

As exploited in Section 3.1, one sticking point for greedy path removal algorithms is the fact that the width of a graph can increase after an edge is fully decomposed. We now identify a new class of graphs, in which the graph does not increase its width during the execution of the algorithm. We show that greedy-weight decomposes “enough” flow at each step in these graphs, giving a $O(\log \text{Val}(X))$ -approximation for $\text{MFD}_{\mathbb{N}}$.

If $X \geq 0$ is a flow on a DAG G , we write $G|_X$ (G restricted to X) to mean the spanning subgraph of G made up of the edges $e \in E$ such that $X(e) \neq 0$. Conversely, if S is a subgraph of G , we write $X|_S$ (X restricted to S) to mean the pseudo-flow X only on the edges of S . In the case of $\text{MFD}_{\mathbb{N}}$, once an edge is fully decomposed, it cannot be used in future paths, possibly increasing the width of the graph that can be used to decompose the rest of the flow and sometimes triggering an increase of the size of a minimum flow decomposition as well. We call a graph *width-stable* if it does not have this issue.

Definition 11 (Width-stable Graph). We say that a graph G is *width-stable* if, for any non-negative flows $X \leq Y$ on G , it holds that $\text{width}(G|_X) \leq \text{width}(G|_Y)$.

Many useful $\text{MFD}_{\mathbb{N}}$ instances satisfy Definition 11. For example, the first proof of MFD ’s NP-hardness [Vatinen et al. 2008] was a reduction to a very simple graph of this form, as shown in Figure 1; this means that $\text{MFD}_{\mathbb{N}}$ restricted to width-stable graphs is also NP-hard.

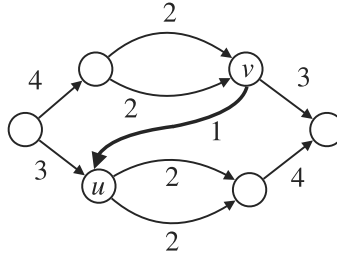


Fig. 6. $\text{MFD}_{\mathbb{N}}$ instance with $\text{width}(G) = 3$, $\text{Val}(X) = 7$, $\text{Val}(X)/\text{width}(G) = 7/3 > 2$, but no path can carry more flow than 2. By Lemma 13, this is equivalent to G not being width-stable, there exists an s - t path saturating the central edge from v to u of weight 1, making the graph a funnel and increasing $\text{width}(G)$ to 4.

Definition 12 (Garlet Millani et al. [2020]). We call an s - t DAG G *funnel* if every s - t path has a private edge that is not contained in any other s - t path.

Funnels are simple graphs in the sense that they admit a unique flow decomposition [Khan et al. 2022]. We use funnels in Lemma 13 to characterize graphs that are not width-stable. Funnels generalise in/out-forests: along any s - t path nodes v first satisfy $\deg^-(v) \leq 1 \leq \deg^+(v)$ and then $\deg^-(v) \geq 1 \geq \deg^+(v)$. We call a node v *forking* (resp. *merging*) if $\deg^+(v) > 1$ (resp. if it is $\deg^-(v) > 1$). For a funnel subgraph F of G we call a path in G from a merging node in F to a forking node in F a *central path* of the funnel. The graphs $(G_\ell, X_{\ell,B})$ in Section 3.1 are precisely funnels with central paths.

The next property that we need is that there is always, during the execution of the greedy-weight algorithm, a path carrying “enough” flow from s to t .

LEMMA 13. Let G be an s - t DAG. The following statements are equivalent:

- (1) G is width-stable,
- (2) G has paths of large weight: for any flow $X \geq 0$ on G , there exists an s - t path in $G|_X$ carrying $\text{Val}(X)/\text{width}(G|_X)$ flow,
- (3) G has no funnel subgraph with a central path.

See Figure 6 for an example.

PROOF. (1) $\Rightarrow$ (2): Let $G = (V, E)$ be an s - t DAG with flow $X \geq 0$ for which there is no s - t path in $G|_X$ carrying $b = \text{Val}(X)/\text{width}(G|_X)$ flow. We will show that G is not width-stable. For simplicity, we assume $G = G|_X$, the result follows immediately for any supergraph of G . Let S be the set of vertices reachable from s by a path carrying b flow. By assumption, $t \notin S$, so $(S, V \setminus S)$ is an s , t -cut and $X(u, v) < b$ for any edge (u, v) with $u \in S, v \in V \setminus S$. Define T to be the set of vertices that can reach t via a path involving vertices only from $V \setminus S$:

$$T := V \setminus \{v \in V \mid \text{all } v-t \text{ paths cross } S\}.$$

Since $S \subseteq V \setminus T$, $(V \setminus T, T)$ is an s , t -cut. Note that if (u, v) is an edge with $u \in V \setminus T$ and $v \in T$, then also $u \in S$ and $v \in V \setminus S$ and thus $X(u, v) < b$. The set C defined by

$$C := \{(u, v) \in E \mid u \in V \setminus T, v \in T\} = E \cap (S \times T)$$

is an s , t -cut set (a set of edges that every s - t path has to cross) and we have $X(e) < b$ for every $e \in C$. This implies $|C| > \text{width}(G)$.

We construct a new flow $Y \geq 0$ on G for which C is an antichain in $G|_Y$. Initially, $Y := X$. Define the following paths:

- $p_s(u)$ for all $u \in S$: an s - u path with all nodes in S ,
- $p_t(v)$ for all $v \in T$: a v - t path with all nodes in T .

We keep the **invariant** that the paths exist and do not change in $G|_Y$ throughout the construction of Y . Let $b(v, u)$ denote a path from $v \in T$ to $u \in S$, all of whose internal nodes are from $V \setminus (S \cup T)$, and note that such a path exists iff C is not an antichain. Assume it carries flow $\mu > 0$ (i.e., μ is the minimum flow value along $b(v, u)$), we then perform the following operation on Y :

$$Y(e) = \begin{cases} Y(e) - \mu & \text{if } e \in b(v, u) \\ Y(e) + \mu & \text{if } e \in p_s(u) \cup p_t(v). \end{cases}$$

Note that after the operation Y remains a flow and that none of the three paths have pairwise intersecting edges. The process does not violate the **invariant** and repeating eventually it destroys all paths from T to S , making C an antichain in $G|_Y$. This shows that G is not width-stable: $Y \leq X + Y$ and $\text{width}(G|_Y) > \text{width}(G|_{X+Y}) = \text{width}(G)$.

(2) $\Rightarrow$ (3): Assume that G has a funnel subgraph F with a central path P . Let C be a maximum antichain of F , and note that every maximum antichain of a funnel consists of private edges only. We define flows Z_{stable} , Z_{unstable} , and Z :

- Z_{stable} is defined to be the flow induced³ by the minimum path cover of F^4 (i.e., $G|_{Z_{\text{stable}}} = F$).
- Z_{unstable} is defined to be the flow induced by the following path cover of the graph consisting of F and P : One s - t path goes along P , covering two edges in C ,⁵ and the other paths cover F avoiding P , this is possible with additional $|C| - 2$ paths.
- Finally, $Z := Z_{\text{stable}} + Z_{\text{unstable}}$.

We have $\text{Val}(Z) = \text{Val}(Z_{\text{stable}}) + \text{Val}(Z_{\text{unstable}}) = |C| + (|C| - 1) = 2|C| - 1$ and $\text{width}(G|_Z) \leq |C| - 1$, and thus

$$\frac{\text{Val}(Z)}{\text{width}(G|_Z)} \geq \frac{2|C| - 1}{|C| - 1} > 2,$$

but all s - t paths in G carry no more flow than 2, because C is an s , t -cut set of G and all Z flow values on C are 2.

(3) $\Rightarrow$ (1): Assume that $G = (V, E)$ is not width-stable, let $Y \geq X \geq 0$ be flows on G with $\text{width}(G|_X) > \text{width}(G|_Y)$ and let C' be a maximum antichain of $G|_X$. Let F be a funnel subgraph of $G|_X$ containing C' as maximum antichain, and let C be the rightmost maximum antichain of F (that is, the head of every edge in C is t or is merging). Since C is not an antichain of $G|_Y$, there must be a path P in $G|_Y$ connecting two edges in C , and it starts at a merging node. It must also enter a forking node, because otherwise adding the path would not decrease the width. This shows that a prefix path of P is a central path of F . $\square$

LEMMA 14. *Let $G = (V, E)$ be a width-stable graph, $\text{width}(G) \geq 2$. Greedy-weight uses at most $\lceil \log \text{Val}(X) / \log \frac{\text{width}(G)}{\text{width}(G)-1} \rceil + 1$ paths to decompose any flow $X : E \rightarrow \mathbb{N}$.*

PROOF. Let $b = \text{width}(G)$. Since G is width-stable, greedy-weight removes a path of weight at least $\text{Val}(X')/b$ at every step by Lemma 13, where X' is the remaining flow of the corresponding step. As such, after c steps $\text{Val}(X') \leq \text{Val}(X)(\frac{b-1}{b})^c$. If $\text{Val}(X)(\frac{b-1}{b})^c < 1$, then $\text{Val}(X') = 0$, since $\text{Val}(X)$ and the weights of the removed paths belong to $\mathbb{N}$. Solving for c we obtain $c >$

³Recall that the induced flow X of a path cover $(P_i)_1^{\ell}$ is defined as $X = P_1 + \dots + P_{\ell}$, where we identify each path P_i with a 0/1-flow with value 1 on the path edges (i.e., precisely the characteristic function of P_i).

⁴Since funnels admit a unique flow decomposition, they also admit a unique minimum s - t path cover.

⁵Indeed, this path covers *exactly* two edges in C : one edge needs to be covered to reach P and another edge must be covered to reach t , and since G is a DAG, we cannot use the same edge twice.

$\log \text{Val}(X)/\log \frac{b}{b-1}$. Therefore, greedy-weight takes (uses) at most $c = \lfloor \log \text{Val}(X)/\log \frac{b}{b-1} \rfloor + 1$ steps (paths). $\square$

THEOREM 2. *Let $G = (V, E)$ be a width-stable graph and $X : E \rightarrow \mathbb{N}$ a flow. Greedy-weight is a $O(\log \text{Val}(X))$ -approximation for $\text{MFD}_{\mathbb{N}}$ on (G, X) .*

PROOF. Assume $X > 0$ (otherwise, replace G by $G|_X$). Thus, $b = \text{width}(G) \leq \text{mfd}_{\mathbb{N}}(G, X)$, since any flow-decomposition of X induces a path cover of E . If $b \leq 1$ greedy-weight finds an optimal solution. Otherwise $b \geq 2$, and Lemma 14 implies that greedy-weight is a $O(\frac{\log \text{Val}(X)}{b \log \frac{b}{b-1}}) = O(\log \text{Val}(X))$ -approximation for $\text{MFD}_{\mathbb{N}}$ ($b \log \frac{b}{b-1} = O(1)$ for $b \geq 2$). $\square$

Finally, we show that series-parallel graphs are width-stable, and thus greedy-weight is a $O(\log \text{Val}(X))$ -approximation on them.

Definition 15 (Series-parallel Graph [Eppstein 1992]). A graph is a *two-terminal series-parallel* (series-parallel for short) graph with terminal nodes s and t if:

- it consists of a single edge directed from s to t , and no other nodes, or
- it can be obtained from two (smaller) two-terminal series-parallel graphs G_1 and G_2 , with terminal nodes s_1, t_1 , and s_2, t_2 , respectively, by either
 - identifying $s = s_1 = s_2$ and $t = t_1 = t_2$ (*parallel composition* of G_1 and G_2), or
 - identifying $s = s_1, t_1 = s_2$, and $t = t_2$ (*series composition* of G_1 and G_2).

COROLLARY 16. *Greedy-weight is a $O(\log \text{Val}(X))$ -approximation for $\text{MFD}_{\mathbb{N}}$ on series-parallel graphs.*

PROOF. Using Theorem 2, it remains to prove that any series-parallel graph $G = (V, E)$ with any flow $X : E \rightarrow \mathbb{N}$ is width-stable. We prove it using structural induction. The base case is when G is single edge from s to t , and they are trivially width-stable.

Suppose now that G is obtained by the composition of series-parallel graphs G_1, G_2 , and let $X \leq Y$ by any non-negative flows on G . Let $X_i = X|_{G_i}, Y_i = Y|_{G_i}$ and let x_i denote $\text{width}(G_i|_{X_i}), y_i$ denote $\text{width}(G_i|_{Y_i})$, for $i = 1, 2$. $x_i \leq y_i$ for $i = 1, 2$.

If the composition operation is parallel composition, then $\text{width}(G) = \text{width}(G_1) + \text{width}(G_2)$ (since edges of G_1 cannot reach edges of G_2 , and vice versa) and $\text{width}(G|_X) = x_1 + x_2$, $\text{width}(G|_Y) = y_1 + y_2$ (since $G|_X$ and $G|_Y$ are also series-parallel), and G is width-stable by the inductive hypothesis that $x_i \leq y_i$, for $i = 1, 2$.

If the composition operation is series composition, width-stability follows analogously to the parallel composition by replacing sum with maximum. $\square$

However, note that there are width-stable graphs that are not series-parallel.

4 WIDTH HELPS SOLVE $\text{MCCD}_{\mathbb{Z}}$

In this section, we give an approximation algorithm for $\text{mfd}_{\mathbb{Z}}(G, X)$. We will obtain this for a more general problem variant, which can be defined as follows. We are given directed a graph $G = (V, E, c)$ with no sink or source nodes, with (cost) a function $c : E \rightarrow \mathbb{R}_{\geq 0}$. The *cost* of a circulation X is defined as $\tilde{c}(X) = \sum_{e \in E} c(e)X(e)$. Note that $\tilde{c}(\cdot)$ is a linear function: $\tilde{c}(X + Y) = \tilde{c}(X) + \tilde{c}(Y)$ for any two circulations X, Y on G .

Definition 17. Given (G, X) of a graph $G = (V, E, c)$ and a circulation $X : E \rightarrow \mathbb{Y}$, a *circulation decomposition* of size k of (G, X) is a family of circulations $Y_i : E \rightarrow \mathbb{N}$ with weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $X = w_1 Y_1 + \dots + w_k Y_k$. We call the problem of finding a circulation decomposition

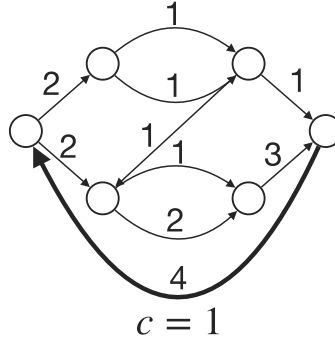


Fig. 7. Reduction in Lemma 18 of a graph with total flow 4 from $\text{MFD}_{\mathbb{Y}}$ to $\text{MCCD}_{\mathbb{Y}}$. The bold edge is the addition to the DAG and is the only edge with cost 1 while all other edges have cost 0.

of minimum cost $\tilde{c}(Y_1 + \dots + Y_k)$ the *Minimum Cost Circulation Decomposition* or $\text{MCCD}_{\mathbb{Y}}$ and we write $\text{mccd}_{\mathbb{Y}}(G, X)$ for the minimum cost.

Decomposing into *non-negative* weighted circulations rather than paths is a natural generalisation, as paths can also be seen as flows with value 1 along the path. The following reduction (Figure 7) shows that $\text{MFD}_{\mathbb{Y}}$ can be regarded as a special case of $\text{MCCD}_{\mathbb{Y}}$.

LEMMA 18. $\text{MCCD}_{\mathbb{Y}}$ is NP-hard.

PROOF. Given an s - t DAG $G = (V, E)$ and flow $X : E \rightarrow \mathbb{Y}$, we define a graph $G' = (V, E', c)$ with $E' = E \cup \{(t, s)\}$ and $c : E' \rightarrow \{0, 1\}$, $c(u, v) = 1 \iff (u, v) = (t, s)$, that is, cost 1 only for the edge (t, s) . Let $X' : E' \rightarrow \mathbb{Y}$, $X'(e) = X(e)$ for $e \in E$ and $X'(t, s) = \text{Val}(X)$ be a circulation on G' . The cost of a MCCD $(Y'_1, w'_1), \dots, (Y'_k, w'_k)$ of (G', X') is equal to $\tilde{c}(Y'_1) + \dots + \tilde{c}(Y'_k) = Y'_1(t, s) + \dots + Y'_k(t, s)$. Note that an s - t path $P : E \rightarrow \{0, 1\}$ yields a circulation $Y_P : E' \rightarrow \{0, 1\}$ of cost 1, which implies $\text{mfd}_{\mathbb{Y}}(G, X) \geq \text{mccd}_{\mathbb{Y}}(G', X')$. Define flows $Y_i : E \rightarrow \mathbb{Y}$ with $Y_i(e) = Y'_i(e)$ for $i = 1, \dots, k$. Decomposing each flow Y_i trivially into $\text{Val}(Y_i) = Y'_i(t, s)$ paths and assigning them weights w'_i yields a Flow Decomposition of (G, X) , showing $\text{mccd}_{\mathbb{Y}}(G', X') \geq \text{mfd}_{\mathbb{Y}}(G, X)$, and thus this Flow Decomposition is minimum. $\square$

Definition 19. Given $S \subseteq E$ and a graph G , we call a circulation C of minimum cost satisfying $C(e) \geq 1$ for all $e \in S$ Minimum Cost Circulation Cover of S , and we write $\text{mccc}_S(G) = \tilde{c}(C)$. If $S = E$, we use $\text{mccc}(G)$ instead.

Note that since every circulation decomposition of a graph G covers the edges of non-zero circulation, $\text{mccc}_S(G) \leq \text{mccd}_{\mathbb{Y}}(G, X)$ with $S = \{e \in E \mid X(e) \neq 0\}$. This generalises the width lower bound of $\text{MFD}_{\mathbb{Y}}$ to $\text{MCCD}_{\mathbb{Y}}$. Given an s - t DAG G , $\text{width}(G) = \text{mccc}(G')$ for the graph G' obtained by the reduction in Lemma 18. We will need a Minimum Cost Circulation Cover for our approximation approach:

LEMMA 20 (GABOW AND TARJAN [1989], THEOREM 3.6). *Let $G = (V, E, c)$ be a graph, and $S \subseteq E$. A Minimum Cost Circulation Cover of S can be computed in $O(n \log m(m + n \log n))$ time.*

The idea behind our approximation algorithm for $\text{MCCD}_{\mathbb{Z}}$ is that a circulation $X : E \rightarrow \mathbb{Z}$ on a graph G can always be decomposed into circulations of total cost $(\lceil \log \|X\| \rceil + 1) \cdot \text{mccc}(G)$. We show this using two key facts: first, that X can be decomposed into $(\lceil \log \|X\| \rceil + 1)$ circulations with a particular structure, and, second, that each of these circulations can be further decomposed into circulations of total cost of at most $\text{mccc}(G)$. A key step in proving both these facts is a sub-routine which, given an input circulation X , finds another circulation Y with values from $\{0, \pm 1\}$

only (a *unitary* circulation) that matches the parity of X on all edges. Intuitively, given an input circulation X , such a unitary circulation Y can be added to X to “fix” its odd edges to be even, with only a small change to X .

LEMMA 21. *For any circulation $X : E \rightarrow \mathbb{Z}$ on $G = (V, E, c)$, there exists a circulation $Y : E \rightarrow \mathbb{Z}$ such that $X \equiv_2 Y$ and $\|Y\| \leq 1$.*

PROOF. Consider the undirected graph $G_{\text{odd}} = (V, E_{\text{odd}})$, where $E_{\text{odd}} = \{(u, v) \mid (u, v) \in E \text{ and } X(u, v) \text{ is odd}\}$.

Notice that every node of G_{odd} has even degree due to the conservation of flow. Thus, G_{odd} can be written as the edge-disjoint union of cycles. Assign an arbitrary orientation to each cycle and let E_{odd}^+ be the set of edges oriented in this way. Define

$$Y(u, v) = \begin{cases} +1 & \text{if } (u, v) \in E_{\text{odd}}^+ \\ -1 & \text{if } (v, u) \in E_{\text{odd}}^+ \\ 0 & \text{if } \{u, v\} \notin E_{\text{odd}} \end{cases}$$

Notice that Y is a circulation decomposed as a sum of circulations, each along one of the edge-disjoint cycles. Moreover, $X \equiv_2 Y$ and $\|Y\| \leq 1$ by construction. $\square$

Repeatedly applying Lemma 21 and dividing the resulting even circulation by 2, we obtain the first key ingredient of the approach.

COROLLARY 22. *Any (non-zero) circulation $X : E \rightarrow \mathbb{Z}$ can be written as $X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i$, where $Y_i : E \rightarrow \mathbb{Z}$ is a circulation with $\|Y_i\| \leq 1$ for all i .*

PROOF. If $\|X\| \leq 1$ the result follows. Otherwise apply Lemma 21 to obtain Y_0 such that $X \equiv_2 Y_0$ and $\|Y_0\| \leq 1$. Since $X \equiv_2 Y_0$, we can define $X' = (X - Y_0)/2$, and thus $X = 2X' + Y_0$. Recursively repeat this procedure on X' until $\|X'\| \leq 1$, obtaining $Y_0, \dots, Y_k = X'$, so that $X = \sum_{i=0}^k 2^i \cdot Y_i$. Finally, note that at each repetition, $\|X\|$ decreases to at most $\lceil \|X\|/2 \rceil$, thus $k \leq \lceil \log \|X\| \rceil$. $\square$

The following result is the second key ingredient of our approach. It guarantees that any unitary circulation can be decomposed into two circulations of total cost of at most $\text{mccc}(G)$ (see Figure 8 for an example). This is by no means obvious since, among other problems, a unitary circulation may contain positive and negative values which merge and cancel each other out (as in Figure 8(b)).

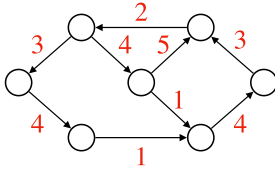
LEMMA 23. *For any circulation $X : E \rightarrow \mathbb{Z}$, $\|X\| \leq 1$, there exist circulations $A, B : E \rightarrow \mathbb{Z}$ such that:*

- (1) $A, B \geq 0$
- (2) $X = A - B$
- (3) $\tilde{c}(A) + \tilde{c}(B) \leq \text{mccc}(G)$

PROOF. Take C such that $C \geq 1$ and $\tilde{c}(C) = \text{mccc}(G)$, according to Lemma 20. Take D such that $D \equiv_2 X + C$ and $\|D\| \leq 1$, according to Lemma 21. Also, assume $\tilde{c}(D) \geq 0$ without loss of generality (otherwise, take $-D$, which satisfies the same properties).

Since $D \equiv_2 X + C$, we have $C - D \pm X \equiv_2 0$. We now consider the circulations $A := (C - D + X)/2$ and $B := (C - D - X)/2$, which have the following properties:

- (1) Notice that $C - D \pm X \geq C - 2$ since $\|D\|, \|X\| \leq 1$. So, $C - D \pm X \geq -1$, since $C \geq 1$. But $C - D \pm X \equiv_2 0$ so $C - D \pm X \geq 0$, therefore $A, B \geq 0$.
- (2) $A - B = \frac{C - D + X}{2} - \frac{C - D - X}{2} = X$.



(a) Edge costs

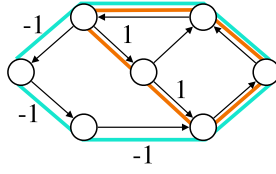
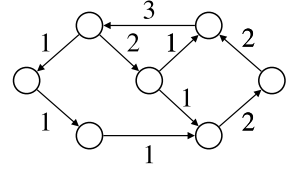
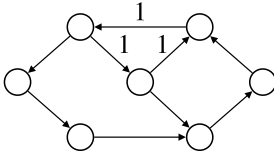
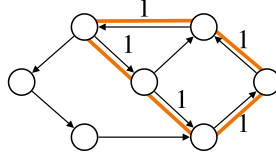
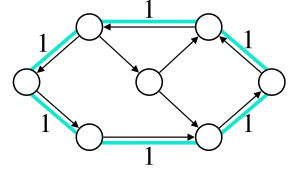
(b) Unitary circulation X on a graph G and a decomposition into two non-negative circulations A and B , of weight 1 in orange (see (e)), and of weight -1 in blue (see (f)), respectively.(c) Circulation C covering all edges of G , of cost $\tilde{c}(C) = \text{mccc}(G) = 42$ (Lemma 20).(d) Unitary circulation D of cost 11 matching the parity of $X + C$, i.e., $D \equiv_2 X + C$ (Lemma 21).(e) Circulation $A = (C - D + X)/2$ of cost 14.(f) Circulation $B = (C - D - X)/2$ of cost 17.

Fig. 8. Example of Lemma 23 applied to a unitary circulation X on a graph G (for clarity, 0 circulation values are not shown). Non-negative circulations A and B can be constructed so that $\tilde{c}(A) + \tilde{c}(B) \leq \text{mccc}(G)$ holds. We obtain a decomposition of X by $X = A - B$.

$$(3) \quad \tilde{c}(A) + \tilde{c}(B) = \tilde{c}(A + B) = \tilde{c}\left(\frac{C-D+X}{2} + \frac{C-D-X}{2}\right) = \tilde{c}(C - D) = \tilde{c}(C) - \tilde{c}(D) \leq \tilde{c}(C) \text{ since } \tilde{c}(D) \geq 0, \text{ and } \tilde{c}(C) = \text{mccc}(G). \quad \square$$

Finally, expressing any circulation as a sum of at most $\lceil \log \|X\| \rceil + 1$ unitary circulations (Corollary 22), and decomposing each unitary circulation into two circulations with cost of at most $\text{mccc}(G)$ (Lemma 23), we can decompose the circulation into circulations of total cost no more than $(\lceil \log \|X\| \rceil + 1) \cdot \text{mccc}(G)$ whose weights are positive and negative powers of two.

THEOREM 24. *Given a graph $G = (V, E, c)$ and a circulation $X : E \rightarrow \mathbb{Z}$ with $k := \lceil \log \|X\| \rceil$, there exist circulations A_i, B_i for $i = 0, \dots, k$ and weights $\{w_0, \dots, w_k\} \subseteq \{2^i \mid i \in \mathbb{N}\}$, with $\tilde{c}(A_0 + \dots + A_k + B_0 + \dots + B_k) \leq (k + 1) \cdot \text{mccc}(G)$ such that $X = w_0(A_0 - B_0) + \dots + w_k(A_k - B_k)$.*

PROOF. Combine Corollary 22 and Lemma 23, getting

$$X = \sum_{i=0}^k 2^i \cdot Y_i = \sum_{i=0}^k 2^i \cdot (A_i - B_i)$$

where $\tilde{c}(A_i + B_i) \leq \text{mccc}(G)$. $\square$

The proof of Theorem 24 suggests a straightforward algorithm for $\text{MCCD}_{\mathbb{Z}}$, which we detail in Algorithm 2 and describe at a high level here. First, iteratively decompose X , yielding $\log \lceil \|X\| \rceil + 1$ unitary circulations. Then use Lemma 23 to decompose each into two circulations of cost at most $\text{mccc}(G)$. However, $\text{mccc}(G)$ is not necessarily a lower bound on $\text{MCCD}_{\mathbb{Z}}$ if the circulation is 0 on some edges, and thus this approach does not directly derive an approximation. To overcome this issue, we instead find a circulation decomposition of a spanning subgraph G' of G for which $\text{mccc}(G')$ lower bounds $\text{mcc}_{\mathbb{Z}}(G, X)$. Namely, we first find a minimum cost circulation cover in G of the subset S of edges with non-zero flow in $O(n \log m(m + n \log n))$ time (according to Lemma 20),

ALGORITHM 1: UNITARY(G, X): Produces a Unitary Circulation Y from an Input Circulation X Such that $X \equiv_2 Y$, as in Lemma 21

```

1:  $E_{odd} \leftarrow$  odd edges of  $G$ , undirected
2:  $C \leftarrow$  a decomposition of  $G_{odd} = (V, E_{odd})$  into cycles, oriented arbitrarily
3:  $E_{odd}^+ \leftarrow$  directed edges of  $C$ 
4: for  $(u, v) \in E$  do
5:   if  $(u, v) \in E_{odd}^+$  then
6:      $Y(u, v) \leftarrow +1$ 
7:   else if  $(v, u) \in E_{odd}^+$  then
8:      $Y(u, v) \leftarrow -1$ 
9:   else
10:     $Y(u, v) \leftarrow 0$ 
11:   end if
12: end for
13: return  $Y$ 

```

ALGORITHM 2: Finds The Circulation Decomposition of Theorem 24

```

1: Compute a minimum cost circulation cover of  $\{(u, v) \in E \mid X(u, v) \neq 0\}$  {Lemma 20}
2: Remove from  $G$  any edge not covered by this circulation cover to obtain  $G'$ 
3:  $\mathcal{P} \leftarrow []$ ,  $\mathcal{W} \leftarrow []$  {length-zero vectors}
4:  $C \leftarrow$  circulation of cost  $mccc(G')$ ,  $C \geq 1$  {Lemma 20}
5:  $D \leftarrow$  UNITARY( $G', C$ ); if  $\text{Val}(D) < 0$  set  $D = -D$  {Algorithm 1}
6:  $i \leftarrow 0$ 
7: while  $\|X\| > 1$  do
8:    $Y_i \leftarrow$  UNITARY( $G', X$ ) {Algorithm 1}
9:    $X \leftarrow (X - Y_i)/2$ 
10:   $i \leftarrow i + 1$ 
11: end while
12:  $Y_i \leftarrow X$ 
13: for  $j \in \{0, \dots, i\}$  s.t.  $Y_j \neq 0$  do
14:    $A \leftarrow C - D + Y_j$ ,  $B \leftarrow C - D - Y_j$ 
15:   Concatenate  $A$  and  $B$  to  $\mathcal{P}$ 
16:   Concatenate  $2^j$  and  $-2^j$  to  $\mathcal{W}$ 
17: end for
18: return  $(\mathcal{P}, \mathcal{W})$ 

```

and then remove from G any edge not covered by the circulation, obtaining G' . By construction, the cost of this circulation cover is a lower bound of $mccd_{\mathbb{Z}}(G, X)$. Moreover, the cost of this circulation cover is exactly $mccc(G')$, since every circulation cover of G' is also a circulation cover of S in G .

To prove the correctness of Algorithm 2, we first define a subroutine implementing Lemma 21.

LEMMA 25. *Algorithm 1 returns a unitary circulation from an input circulation Y such that $X \equiv_2 Y$, as in Lemma 21, in $O(m)$ time.*

PROOF. The correctness of the algorithm is given by Lemma 21. Finally, the first 3 subroutines as well as the entire for-loop takes $O(m)$ time. $\square$

THEOREM 3. *$MCCD_{\mathbb{Z}}$ can be approximated with a factor of $\log \lceil \|X\| \rceil + 1$ in runtime $O(n \log m(m + n \log n) + m \log \|X\|)$.*

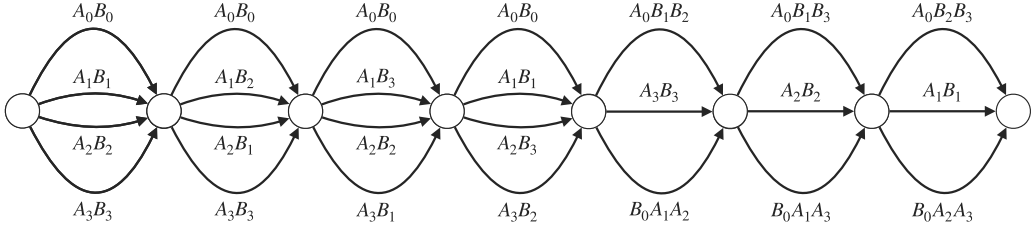


Fig. 9. Paths A_i and B_i ($i \in \{0, 1, 2, 3\}$), each edge being labeled with the paths it appears in. Assign to each path A_i weight a_i , and to each path B_i weight b_i , such that $a_0 = b_0 = 3$, and $a_i = 6^{2i} + 1$ and $b_i = 6^{2i+1} + 5$ for $i = 1, 2, 3$. Define the flow X on G as $X = \sum_{i=0}^3 a_i A_i + \sum_{i=0}^3 b_i B_i$. Note that these weights are a solution of k -FWA $_{\mathbb{N}}$ on input (G, X) with given paths A_i, B_i ($i \in \{0, 1, 2, 3\}$).

PROOF. By Theorem 24 and our previous discussion, Algorithm 2 returns a circulation decomposition for X with no more cost than $(\lceil \log \|X\| \rceil + 1) \cdot \text{mccc}(G') \leq (\lceil \log \|X\| \rceil + 1) \cdot \text{mcccd}_{\mathbb{Z}}(G, X)$. We analyse the runtime line by line. Lines 1 and 4 take $O(n \log m(m + n \log n))$ time by Lemma 20. The call to Algorithm 1 on line 5 takes $O(m)$ time by Lemma 25, and checking the cost of D and flipping signs (if necessary) also takes $O(m)$ time. By Corollary 22, the while loop on line 7 executes at most $\log \lceil \|X\| \rceil + 1$ times, meaning that the entire execution takes $O(m \log \|X\|)$ time since line 8 takes $O(m)$ time by Lemma 25. Since there are at most $\log \lceil \|X\| \rceil + 1$ Y_i 's, the for loop on line 13 executes at most $\log \lceil \|X\| \rceil + 1$ times. Each execution of the for-loop finds two circulations of total cost of at most $\text{mccc}(G')$ in $O(m)$ time, so the whole also loop takes $O(m \log \|X\|)$ time. Thus, the overall runtime is $O(n \log m(m + n \log n) + m \log \|X\|)$. $\square$

With the reduction given in Lemma 18, we obtain an approximation algorithm of the same ratio for $\text{MFD}_{\mathbb{Z}}$. However, we can improve the runtime of Lemma 20:

COROLLARY 26. *Algorithm 2 is also a $\log \lceil \|X\| \rceil + 1$ -approximation for $\text{MFD}_{\mathbb{Z}}$ with runtime $O(m(n + \text{width}(G) \log \lceil \|X\| \rceil))$.*

PROOF. This is directly achieved by using Theorem 3 with Lemma 18 and by calculating the $\text{width}(G)$ according to Lemma 7. Note that the flows A and B need to be trivially decomposed into at most $\text{width}(G)$ paths, causing the additional factor in the runtime. $\square$

A theorem analogous to Theorem 24 for $\text{MCCD}_{\mathbb{N}}$ is desirable, but cannot be achieved directly with the previous methods, as Lemma 21 makes use of negative weights. However, the approach can be adapted for $\text{MCCD}_{\mathbb{N}}$ if the input flows are width-stable (Definition 11), and if it is possible to “fix” the odd flows to be even with only $\text{mccc}(G)$ unitary flows, which we leave as an open question.

5 SOLVING THE k -FLOW WEIGHT ASSIGNMENT PROBLEM

In this section, we consider a restriction of MFD from Kloster et al. [2018] (see Figure 9 for an example).

Definition 27 (k -Flow Weight Assignment). Given a flow $X : E \rightarrow \mathbb{Y}$ on a graph $G = (V, E)$ and a set of s - t paths $\{P_1, \dots, P_k\}$, the problem of finding an assignment of weights to the paths, such that they form a flow decomposition of (G, X) , is called k -Flow Weight Assignment (k -FWA). We write k -FWA $_{\mathbb{Y}}$ if we require the path weights to belong to $\mathbb{Y}$.

Given k s - t paths, k -FWA can be solved by a linear system defined by $Lw = X$, where $X_j \in \mathbb{Y}$ is equal to the flow $X(e_j)$ of the edge e_j (we identify flows $X : E \rightarrow \mathbb{Y}$ with vectors $X \in \mathbb{Y}^m$) and L is the $m \times k$ 0/1 matrix with $L_{i,j} = 1$ if and only if path P_j crosses edge e_i . The resulting

solution $w \in \mathbb{Y}^k$ is the weight assignment to each path. For a flow graph (G, X) , we denote by $L_{\mathbb{Y}} = L_{\mathbb{Y}}(P_1, \dots, P_k) = \{w \in \mathbb{Y}^k \mid X = \sum_{j=1}^k P_j w_j\}$ the linear system corresponding to the paths $P_1, \dots, P_k$.

We shortly discuss how to solve k -FWA $_{\mathbb{Z}}$. The linear system defined by the paths is a system of linear Diophantine equations. It is well known that integer solutions to such systems can be found in polynomial time; see, for example, [Schrijver 1986, Chapter 5].

Solving k -FWA $_{\mathbb{N}}$ turns out to be more difficult, its the linear system contains the inequality $w \geq 0$. In fact, it was shown [Kloster et al. 2018] that k -FWA $_{\mathbb{N}}$ is NP-hard. The program Toboggan [Kloster et al. 2018] implements a linear FPT algorithm for MFD $_{\mathbb{N}}$ and one step of the algorithm is to solve k -FWA $_{\mathbb{N}}$ using an ILP [Kloster et al. 2018]. The authors state the following conjecture.

CONJECTURE 28 (KLOSTER ET AL. [2018]). *If $(P_1, \dots, P_k)$ are the paths of a minimum flow decomposition of (G, X) , then the linear system $L_{\mathbb{N}}(P_1, \dots, P_k)$ has full rank k .*

In case of a fractional decomposition (in which the weights of the paths are allowed to be rational non-negative numbers), it is indeed true that the induced linear system is of full rank k [Vatinlen et al. 2008]. As mentioned in the introduction, if the conjecture turned out to be true for natural numbers, Toboggan could avoid resorting to solving an ILP, since just solving the standard linear system at hand would return its unique solution. As observed by the authors, this would decrease the asymptotic worst case upper bound of Toboggan.

We show that this conjecture is false using a counterexample. Consider the input for k -FWA $_{\mathbb{N}}$ from Figure 9 and the solution therein. We now give another solution for k -FWA $_{\mathbb{N}}$ on this input, namely the following path weights: $a_0 = 5, b_0 = 1$, and $a_i = 6^{2i} + 2, b_i = 6^{2i+1} + 4$, for $i = 1, 2, 3$. One can easily verify that this is another solution to k -FWA $_{\mathbb{N}}$ on the input in Figure 9, thus proving that the rank of the corresponding linear system is strictly less than 8.

To disprove Conjecture 28, it remains to show that any flow decomposition contains at least 8 paths. Due to the technicality of this proof (and its exhaustive case-by-case analysis), we only explain the intuition behind the construction in Figure 9 and behind the correctness proof. However, as an additional check we also ran both Toboggan [Kloster et al. 2018] and a recently developed ILP solver for MFD $_{\mathbb{N}}$ [Dias et al. 2022] on this instance, both returning $\text{mfd}_{\mathbb{N}}(G, X) = 8$.

The intuition is as follows. The graph can be divided into two parts: the graph induced by the first 5 vertices in topological order (left part) and the one induced by the last 4 (right part). We say that a path is **fixed** if every minimum flow decomposition of the graph contains this path. The paths A_i and B_i have exponentially growing weight for growing i and get shuffled around with different permutations of the paired labels $A_i B_j$ on the left part. Due to the exponential growth, ensuring the correct parity on all edges of the right part, we can fix the paths A_i and B_i for $i = 1, 2, 3$. This allows us to interpret flow decompositions of less than 8 paths as decompositions with 8 paths, where either A_0 or B_0 carries weight 0. Consider a flow decomposition where, we assign two paths of weights λ_1 and λ_2 on the edges labeled $A_0 B_0$. For any $\delta \geq 0$, $(\lambda_1 - \delta) + (\lambda_2 + \delta) = a_0 + b_0$ and equivalently for all other edges on the left part. If we decrease λ_1 by some $\delta > 0$, the weights of B_1, B_2 , and B_3 each increase by $\delta/2$. And thus, δ must be even. Due to the parity of a_0 and b_0 , they can never reach 0.

6 CONCLUSIONS

In this article we have shown for the first time that width, a natural lower bound for MFD, is also useful when investigating its approximability. On the one hand, using width is a key insight in understanding where greedy path removal heuristics fail. On the other hand, graphs where width is well-behaved (e.g., series-parallel graphs) have a guaranteed approximation factor. Moreover,

we generalised MFD to the problem to minimising the cost of a circulation decomposition, and showed that the integer version can be approximated even better by combining parity arguments of unitary circulations and a decomposition of such circulations of cost equal to the minimum cost to cover the graph. Finally, we have corroborated the complexity gap between the positive integer and the full integer case by disproving a conjecture from Kloster et al. [2018] (also motivating the heuristic in Shao and Kingsford [2017]), which would have had sped up their FPT algorithm for $\text{MFD}_{\mathbb{N}}$.

Our results open up new avenues for further research on MFD. For example, can the width help find larger classes of graphs for which some greedy path removal (or even some sort of greedy *path cover* removal) algorithms have a guaranteed approximation factor? Can we get $\Omega(n)$ worst case approximation ratio of greedy-weight for dense graphs without parallel edges? Can the power-of-two decomposition approach be applied with other factors besides two? Can better path cover-like lower bounds help (e.g., path covers which cannot use an edge more times than its flow value, also computable in polynomial time)? How do our algorithms perform in practice?

REFERENCES

- Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. Network flows: Theory, algorithms and applications. New Jersey: Prentice-Hall (1993).
- Jasmijn A. Baaijens, Leen Stougie, and Alexander Schönhuth. 2020. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *Proceedings of the International Conference on Research in Computational Molecular Biology*. Springer, 221–222.
- Elsa Bernard, Laurent Jacob, Julien Mairal, and Jean-Philippe Vert. 2014. Efficient RNA isoform identification and quantification from RNA-Seq data with network flows. *Bioinformatics* 30, 17 (2014), 2447–2455.
- Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. 2013. Robust and adaptive network flows. *Operations Research* 61, 5 (2013), 1218–1242. DOI: <https://doi.org/10.1287/opre.2013.1200>
- Manuel Cáceres, Massimo Cairo, Brendan Mumey, Romeo Rizzi, and Alexandru I. Tomescu. 2022. Sparsifying, shrinking and splicing for minimum path cover in parameterized linear time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms*. 359–376. DOI: <https://doi.org/10.1137/1.9781611977073.18>
- Fernando H. C. Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. 2023. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics* 39, 11 (2023), btad640. <https://doi.org/10.1093/bioinformatics/btad640>
- Fernando H. C. Dias, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. 2022. Fast, flexible, and exact minimum flow decompositions via ILP. In *Proceedings of the Research in Computational Molecular Biology: 26th Annual International Conference*. Springer, 230–245.
- Robert P. Dilworth. 1950. A decomposition theorem for partially ordered sets. *Annals of Mathematics* 51, 1 (1950), 161–166. Retrieved from <http://www.jstor.org/stable/1969503>
- David Eppstein. 1992. Parallel recognition of series-parallel graphs. *Information and Computation* 98, 1 (1992), 41–55. DOI: [https://doi.org/10.1016/0890-5401\(92\)90041-D](https://doi.org/10.1016/0890-5401(92)90041-D)
- Delbert R. Fulkerson. 1956. Note on dilworth’s decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society* 7, 4 (1956), 701–702.
- Harold N. Gabow and Robert E. Tarjan. 1989. Faster scaling algorithms for network problems. *SIAM Journal on Computing* 18, 5 (1989), 1013–1036.
- Marcelo Garlet Millani, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. 2020. Efficient algorithms for measuring the funnel-likeness of DAGs. *Journal of Combinatorial Optimization* 39, 1 (2020), 216–245.
- Thomas Gatter and Peter F. Stadler. 2019. Ryūto: Network-flow based transcriptome reconstruction. *BMC Bioinformatics* 20, 1 (2019), 1–14.
- Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. 2012. How to split a flow?. In *Proceedings of the 2012 IEEE INFOCOM*. IEEE, 828–836.
- Amit Jain and Naga Chandrasekharan. 1993. An efficient parallel algorithm for min-cost flow on directed series-parallel networks. In *Proceedings of the 7th International Parallel Processing Symposium*. 188–192. DOI: <https://doi.org/10.1109/IPPS.1993.262879>
- Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I. Tomescu. 2022. Improving rna assembly via safety and completeness in flow decompositions. *Journal of Computational Biology* 29, 12 (2022), 1270–1287.

- Kyle Kloster, Philipp Kuinke, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villamil, Blair D. Sullivan, and Andrew van der Poel. 2018. A practical fpt algorithm for flow decomposition and transcript assembly. In *2018 Proceedings of the 20th Workshop on Algorithm Engineering and Experiments*. SIAM, 75–86.
- Brendan Mumey, Samareh Shahmohammadi, Kathryn McManus, and Sean Yaw. 2015. Parity balancing path flow decomposition and routing. In *Proceedings of the 2015 IEEE Globecom Workshops*. IEEE, 1–6.
- Nils Olsen, Natalia Kliewer, and Lena Wolbeck. 2022. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time-space network models. *Central European Journal of Operations Research* 30, 3 (2022), 883–919. DOI : <https://doi.org/10.1007/s10100-020-00705-6>
- James B. Orlin. 2013. Max flows in $O(nm)$ time, or better. In *Proceedings of the Symposium on Theory of Computing Conference, Palo Alto, CA, USA, June 1–4, 2013*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM, 765–774. DOI : <https://doi.org/10.1145/2488608.2488705>
- Mihaela Pertea, Geo M. Pertea, Corina M. Antonescu, Tsung-Cheng Chang, Joshua T. Mendell, and Steven L. Salzberg. 2015. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature Biotechnology* 33, 3 (2015), 290–295.
- Alexander Schrijver. 1986. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc.
- Mingfu Shao and Carl Kingsford. 2017. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM transactions on computational biology and bioinformatics* 16, 2 (2017), 658–670.
- Mingxiang Teng, Michael I. Love, Carrie A. Davis, Sarah Djebali, Alexander Dobin, Brenton R. Graveley, Sheng Li, Christopher E. Mason, Sara Olson, Dmitri Pervouchine, et al. 2016. A benchmark for RNA-seq quantification pipelines. *Genome Biology* 17, 1 (2016), 1–12.
- Alexandru I. Tomescu, Travis Gagie, Alexandru Popa, Romeo Rizzi, Anna Kuosmanen, and Veli Mäkinen. 2015. Explaining a weighted DAG with few paths for solving genome-guided multi-assembly. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12, 6 (2015), 1345–1354.
- Alexandru I. Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. 2013. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC bioinformatics* 14, 5 (2013), S15:1–S15:10.
- Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. 1982. The recognition of series parallel digraphs. *SIAM Journal on Computing* 11, 2 (1982), 298–313. DOI : <https://doi.org/10.1137/0211023> arXiv:<https://doi.org/10.1137/0211023>
- Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. 2008. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research* 185, 3 (2008), 1390–1401.
- Lucia Williams, Gillian Reynolds, and Brendan Mumey. 2019. RNA transcript assembly using inexact flows. In *Proceedings of the 2019 IEEE International Conference on Bioinformatics and Biomedicine*. IEEE, 1907–1914.

Received 29 April 2023; revised 23 December 2023; accepted 14 January 2024

Paper III

Andreas Grigorjew, Wanchote Jiamjitrak, Brendan Mumey, and Alexandru I. Tomescu

Parameterised Approximation and Complexity of Minimum Flow Decompositions

Submitted.

Copyright © Andreas Grigorjew, Wanchote Jiamjitrak, Brendan Mumey, and Alexandru I. Tomescu.

III

Parameterised Approximation and Complexity of Minimum Flow Decompositions

Andreas Grigoriou¹, Wanchote Jiamjitrak¹, Brendan Mumey², and Alexandru I. Tomescu¹

¹Department of Computer Science, University of Helsinki

²School of Computing, Montana State University

Abstract

Minimum flow decomposition (MFD) is the strongly NP-hard problem of finding a smallest set of integer weighted paths in a graph G whose weighted sum is equal to a given flow f on G . Despite its many practical applications, we lack an understanding of graph structures that make MFD easy or hard. In particular, it is not known whether a good approximation algorithm exists when the weights are positive.

On the positive side, the main result of this paper is that MFD can be approximated within a factor $O(\log \|f\|)$ (where $\|f\|$ is the largest flow weight of all edges) times the ratio between the *parallel-width* of G (introduced by Deligkas and Meir, MFCS 2018) and the *width* of G (minimum number of paths to cover all edges). In particular, when the MFD size is at least the parallel-width of G , this becomes the first parameterised $O(\log \|f\|)$ -factor approximation algorithm for MFD over positive integers. We also show that there exist instances where the ratio between the parallel-width of G and the MFD size is arbitrarily large, thus narrowing down the class of graphs whose approximation is still open. We achieve these results by introducing a new notion of *flow-width* of (G, f) , which unifies both the width and the parallel-width and may be of independent interest.

On the negative side, we show that small-width graphs do not make MFD easy. This question was previously open, because width-1 graphs (i.e. paths) are trivially solvable, and the existing NP-hardness proofs use graphs of unbounded width. We close this problem by showing the tight results that MFD remains strongly NP-hard on graphs of width 3, and NP-hard on graphs of width 2 (and thus also parallel-width 2). Moreover, on width-2 graphs (and more generally, on constant parallel-width graphs), MFD is solvable in quasi-polynomial time on unary-coded flows.

1 Introduction

Minimum Flow Decomposition (MFD) of a given flow on a directed graph is the problem of finding a minimum sized set of weighted paths, such that for every edge, the sum of the weights of the paths crossing this edge is equal to the flow on the edge. It is a standard result that every flow on a directed acyclic graph can be decomposed to at most $m - n + 2$ weighted paths [1, 24], where n is the number of nodes and m is the number of edges. In this paper, all graphs will be directed acyclic graphs (DAGs) that are multigraphs (that is, we allow parallel edges) with single source s and single sink t , and we use the term *flow network* to refer to a pair (G, f) of a DAG G and a flow f on G . We also distinguish between two MFD variants: $\text{MFD}_{\mathbb{N}}$ where the flow and path weights are non-negative integers, and $\text{MFD}_{\mathbb{Z}}$ where the flow and path weights can also be negative; we use MFD to refer to both variants and we write $\text{mfd}_{\mathbb{Y}}(G, f)$ for the size of the minimum flow decomposition using path weights in a set $\mathbb{Y}$.

MFD is strongly NP-hard, even on DAGs, and even when the flow values come from $\{1, 2, 4\}$ [13]. However, the problem has a wide range of applications, e.g. in Bioinformatics [3, 8, 11, 19, 22, 21, 25, 2, 20], transportation planning [17] or network design [13, 16, 24]. Despite this widespread use, we lack a good theoretical understanding behind the complexity of $\text{MFD}_{\mathbb{N}}$, and applications just resort to heuristics when solving MFD. For example, there exists no constant-factor (or even $\log m$ factor) approximation algorithm for $\text{MFD}_{\mathbb{N}}$, nor a proof that such algorithms might not exist. Moreover, as opposed to other NP-hard problems which have been extensively studied on restricted classes of graphs, we have only little knowledge of graph classes that make the problem tractable or any other structural properties that make $\text{MFD}_{\mathbb{N}}$ easier. Below, we describe in more detail the current state of the art around MFD.

1.1 Related work

Kloster et al. [15] have shown that MFD is in linear FPT time $2^{O(k^2)} \cdot (m \cdot \log \|f\|)$, where the parameter k is the size of the optimal solution and $\|f\|$ is the maximum norm of f (i.e., the largest flow weight of all edges). $\text{MFD}_{\mathbb{N}}$ also admits polynomially-sized Integer Linear Programming formulations [7, 12]. It is also known that $\text{MFD}_{\mathbb{N}}$ is APX-hard (i.e., for some $\varepsilon > 0$ there is no $(1 + \varepsilon)$ -approximation unless $\text{P} = \text{NP}$) [13], and there exists an approximation algorithm that decomposes all but an ε fraction of the flow with a factor of $O(1/\varepsilon^2)$ [13].

Recent theoretical progress is due to Cáceres et al. [4], who showed that the *width* of the graph, namely, the minimum number of s - t paths to cover all edges, can play an important role in the decomposition of flows. The width is a natural lower bound to MFD, since every flow decomposition is also such a path cover. For example, using width, they improved the approximation factor lower-bound of the widely-used greedy approach for $\text{MFD}_{\mathbb{N}}$ – to iteratively remove the currently highest weighted s - t path in the graph [24, 11, 22, 3, 20] – to $\Omega(m/\log m)$ in the worst case [4]. To obtain this, they exploited the fact that the width can grow exponentially during the process of the greedy algorithm.

Secondly, Cáceres et al. [4] used width to obtain two approximation algorithms for MFD. The first one, working on $\text{MFD}_{\mathbb{Z}}$, has an approximation ratio of $O(\log \|f\|)$. This algorithm follows a “parity fixing” approach: it constructs a unitary flow (that is, a flow with values in $\{-1, 0, 1\}$), which, when subtracted from f , yields a flow that is even everywhere, and they showed that all unitary flows can be decomposed into at most $\text{width}(G)$ paths with weights in $\{-1, 1\}$. The resulting flow can then be divided by two, to repeat the procedure until the flow is zero. To sum up, Cáceres et al. [4] proved that it is possible to express f as

$$f = \sum_{i=0}^{\lceil \log \|f\| \rceil} 2^i f_i, \quad \text{with} \quad \text{mfd}_{\{-1, 1\}}(G, f_i) \leq \text{width}(G). \quad (1)$$

The same parity fixing approach has been used for $\text{MFD}_{\mathbb{N}}$ in [16] to express f as a sum of $O(\log \|f\|)$ positive flows f_i , but with no upper-bound on the minimum decomposition size of each f_i , and instead proved an exponential $L^{\log \|f\|} \cdot \log \|f\|$ approximation factor bound, where L is the length of the longest s - t path in G . However, they experimentally showed that this approach works well for some randomly constructed classes of graphs. They have shown that such parity fixing flows f_i can be found in $\text{MFD}_{\mathbb{N}}$ via finding minimum flows following certain lower- and upper-bound constraints, and left the problem open, whether one can carefully choose such flows to obtain a better theoretical bound. The clear difference of

$\text{MFD}_{\mathbb{Z}}$ to $\text{MFD}_{\mathbb{N}}$ is that in the former variant the width of the graph stays constant: edges do not get saturated in $\text{MFD}_{\mathbb{Z}}$, and thus $\text{width}(G)$ is a lower-bound to the optimal solution of $\text{MFD}_{\mathbb{Z}}$ throughout the whole parity fixing algorithm. As such, it remained open whether some notion of width can be still employed to obtain an approximation algorithm for $\text{MFD}_{\mathbb{N}}$.

To partially address this, Cáceres et al. [4] have proven that restricting the input to *width-stable* graphs, which are defined as s - t DAGs whose width never increases during the process of the algorithm, improves the approximation factor of greedy to $O(\log \text{Val}(f))$, where $\text{Val}(f)$ is defined as the sum of the flow that leaves s . This assumption was necessary because removing weighted paths from a flow will eventually saturate edges, and this can break paths in the minimum path cover and potentially increase the width. A notable example of width-stable DAGs are *series-parallel* DAGs; a numerous amount of NP-hard problems is easier to solve on them, see e.g. [9, 23]. Despite such better approximation factor for greedy on width-stable graphs, MFD remains strongly NP-hard even in their subclass of series-parallel graphs [24].

1.2 Our contributions

As opposed to other NP-hard problems, we have little understanding of what makes MFD easy (to approximate), or hard. As such, in this paper we establish further connections between structural properties or parameters of the graph, and the approximability status of $\text{MFD}_{\mathbb{N}}$, or its intractability status in relation to these parameters. We show how using two parameters, the width of a DAG G and a parameter, the *parallel-width* $\text{par-width}(G)$, recently introduced by Deligkas and Meir [6], can make $\text{MFD}_{\mathbb{N}}$ easier to solve. We also show that, contrary to $\text{MFD}_{\mathbb{Z}}$, the width of the DAG *alone* (without the parallel-width) is not a sufficient parameter for this goal. Specifically, we present the following contributions.

Flow-width as an improved lower-bound for $\text{MFD}_{\mathbb{N}}$. We first present our main tool to analyse the flow network structure, *flow-width*, as follows.

When adapting the parity fixing decomposition from Equation (1) by Cáceres et al. [4] to $\text{MFD}_{\mathbb{N}}$, we are facing the following issues: i) using $\text{width}(G)$ paths to decompose the positive parity fixing flows f_i might not be possible, as we would be forced to oversaturate edges in some cases, ii) the width of the DAG does not necessarily remain a lower-bound to the optimal solution size as edges get saturated during the decomposition, and thus are unavailable in later steps. To address these, we introduce a new notion of width, *flow-width* of a flow network (G, f) , that uses the given flow f on G as upper-bounds for the number of times covering paths can use edges:

Definition 1 (Flow-width). *Let G be a DAG and f be an integral non-negative flow on G . We define the flow-width of G and f , $\text{fw}(G, f)$, as the smallest number of paths satisfying the following properties:*

1. **Covering** *Every edge $e \in E(G)$ with $f(e) > 0$ appears in at least one path, and*
2. **Upper-bounds** *Every edge $e \in E(G)$ appears in at most $f(e)$ paths.*

By definition, it holds that $\text{width}(G) \leq \text{fw}(G, f)$ for all positive flows f , and moreover the flow-width is still a natural lower-bound to the optimal solution of $\text{MFD}_{\mathbb{N}}$.

In relation to Equation (1), Mumey et al. [16] state the question, whether it is possible to choose flows f_i , whose value is as small as possible, such that their decomposition size is small. We answer this question positively for certain classes of graphs by relating their decomposition size to the flow-width:

Theorem 2. *Given an $\text{MFD}_{\mathbb{N}}$ input (G, f) , we can decompose the flow into $\lfloor \log \|f\| \rfloor + 1$ flows f_i , such that*

$$f = \sum_{i=0}^{\lfloor \log \|f\| \rfloor} 2^i f_i \quad \text{and} \quad \text{mfd}_{\{1\}}(G, f_i) = \text{Val}(f_i) \leq \text{fw}(G, f^{(i)}), \quad (2)$$

where $f^{(0)} = f$ and $f^{(i)} = f - \sum_{j=0}^{i-1} 2^j f_j$. This can be done in time $O(nm \log \|f\|)$.

We then address (ii) by analysing the growth of the flow-width throughout the algorithm. As first contribution, we obtain Theorem 3 below by showing that a DAG is width-stable if and only if the

flow-width is monotone in the flow: $\text{fw}(G, f) \leq \text{fw}(G, g)$ for all flows with $f \leq g$ (i.e. $f(e) \leq g(e)$ for all edges e). This improves the approximation factor for $\text{MFD}_{\mathbb{N}}$ on width-stable graphs from $O(\log \text{Val}(f))$ (as in [4]) to $O(\log \|f\|)$:

Theorem 3. *On width-stable DAGs we can approximate $\text{MFD}_{\mathbb{N}}$ given the input (G, f) with ratio $\lfloor \log \|f\| \rfloor + 1$ in runtime $O(n \log \|f\| \cdot (\text{width}(G) + m))$.*

A parameterised approximation algorithm for $\text{MFD}_{\mathbb{N}}$. Second, using flow-width, we parameterise the parity fixing algorithm originally presented by Mumey et al. [16] with the width of G and the parallel-width of G . The parallel-width of G is defined as the size of the largest minimal cut-set of G in the context of a minor notion on s - t DAGs. This notion of directed minors allows removals of edges (u, v) for which the out-degree of u and the in-degree of v are each at least 2, and edges (u, v) can be contracted when it is the only incoming edge of v or the only outgoing edge of u ¹ [6]. Moreover, the parallel-width can be upper-bounded by excluding such directed minors in the DAG. We show that this corresponds exactly to possible edge saturations of flow decompositions, and we can thus use the parallel-width as a parameter in the approximation. We then prove that the flow-width generalises the width and the parallel-width:

Lemma 4. *For all s - t DAGs G , we have*

$$\text{width}(G) = \min\{\text{fw}(G, f) \mid f > 0\}$$

and

$$\text{par-width}(G) = \max\{\text{fw}(G, f) \mid f \geq 0\}.$$

We use this lemma to express the approximation ratio in terms of the largest possible growth of the flow-width of the flow network during the decomposition.

Theorem 5. *Given a flow network (G, f) with $f > 0$, we can approximate $\text{MFD}_{\mathbb{N}}$ with a ratio of $\left(\frac{\text{par-width}(G)}{\text{width}(G)} \cdot (\lfloor \log \|f\| \rfloor + 1)\right)$ in runtime $O(n \log \|f\| \cdot (\text{par-width}(G) + m))$. Moreover, for flow networks with $\text{mfd}_{\mathbb{N}}(G, f) \geq \text{par-width}(G)$, the ratio is $\lfloor \log \|f\| \rfloor + 1$.*

Additionally, we can use this upper-bound to show an improved computational complexity:

Corollary 6. *Let $c \in \mathbb{N}$ be a constant. $\text{MFD}_{\mathbb{N}}$ on (G, f) can be solved in quasi-polynomial time $\|f\|^{O(\log \|f\|)} \cdot (m + \log \|f\|)$ when $\text{par-width}(G) \leq c$ and f is coded in unary.*

Improved hardness results. In Section 5 we further explore the parameterised complexity of $\text{MFD}_{\mathbb{N}}$. It remains NP-hard on width-stable graphs, since the NP-hardness reduction from [24] uses series-parallel graphs, which are a subclass of width-stable graphs [4]. However, their width is not bounded and it remained open whether having a bounded width makes the problem easier. We address this by proving the following.

Theorem 7. *$\text{MFD}_{\mathbb{N}}$ on G is strongly NP-hard even when $\text{width}(G) = 3$.*

Further, this shows that the ratio $\text{par-width}(G)/\text{width}(G)$ can grow arbitrarily large. We use this fact to show that the parity fixing approach obtains, for some class of graphs, an approximation ratio of $\Omega(\|f\|)$. This describes in detail the remaining class of inputs for which no approximation algorithm with good factor (that is, better than very simple decomposition methods) is known. Lastly, we show that all DAGs have $\text{width}(G) = 2$ if and only if they have $\text{par-width}(G) = 2$ and show the following hardness result.

Theorem 8. *$\text{MFD}_{\mathbb{N}}$ on (G, f) is NP-hard when f is coded in binary, even when $\text{width}(G) = 2$.*

¹This notion of edge contractions is also known as butterfly contraction. Note that butterfly contractions are usually used in simple graphs where parallel edges are merged. Here, we use multigraphs and do not merge parallel edges.

2 Preliminaries

By default, graphs $G = (V(G), E(G))$ are assumed to be acyclic and have w.l.o.g. a single source s and a single sink t . DAGs with multiple sources and sinks can be transformed to have one source and one sink, by adding two vertices s and t , and adding an edge from s to each source vertex and adding an edge from each sink vertex to t . We use n and m to denote the number of vertices and edges, respectively, and we denote by $\deg^+(v)$ and $\deg^-(v)$ the out- and in-degree of a vertex v , respectively. We write $G - H$ for two graphs G and $H \subseteq G$ for the subgraph defined by $V(G) \setminus V(H)$ and $E(G) \setminus \{(u, v) \mid u \in V(H) \text{ or } v \in V(H)\}$. A set of edges is called a cut-set, if removing them from the graph disconnects s from t . We denote by $[k]$ the set $\{1, \dots, k\}$. We call functions $f : E(G) \rightarrow \mathbb{Y}$ *pseudo-flows*² on G , where $\mathbb{Y} \in \{\mathbb{N}, \mathbb{Z}\}$. We use the notation $f + g$ and μf to denote the element wise sum of pseudo-flows and scalar multiplication. The value 0 may (depending on context) denote the pseudo-flow that is equal 0 on every edge. We write $f \leq g$ (resp. $f < g$) to denote $f(e) \leq g(e)$ (resp. $f(e) < g(e)$) for every edge $e \in E(G)$ and two pseudo-flows f, g on G .

A *flow* on G is a pseudo-flow whose internal vertices $V \setminus \{s, t\}$ satisfy the flow conservation (incoming flow is equal to the outgoing flow). The sum of two flows $f + g$, the scalar μf and the pseudo-flow 0 are again flows. We denote by the flow value $\text{Val}(f)$ of f the sum of the outgoing flow of s , and we call $f(e)$ the weight of edge e . We call the pair (G, f) of an s - t DAG G and a flow f a flow network. Given an s - t path P , we also denote by P the indicator flow of the path, that is, $P(e) = 1$ for all $e \in P$ and $P(e) = 0$ otherwise. We say that a path P in (G, f) carries weight μ if $f(e) \geq \mu$ for all $e \in P$, and we define that any $v - v$ path carries ∞ weight.

Definition 9. Given a flow $f : E(G) \rightarrow \mathbb{Y}$ on G , a flow decomposition of size k of (G, f) is a family of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ and weights $(w_1, \dots, w_k) \in \mathbb{Y}^k$ such that $f = w_1 P_1 + \dots + w_k P_k$.

Definition 10. For a flow f on G , let $\text{mfd}_{\mathbb{Y}}(G, f)$ be the smallest size of a flow decomposition of (G, f) using weights in $\mathbb{Y}$. We denote by $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$ the problems of finding a flow decomposition of smallest size for $\mathbb{Y} = \mathbb{N}$ and $\mathbb{Y} = \mathbb{Z}$, respectively.

We let $\|f\|$ denote the maximum norm on flows, and write $f \equiv_2 g$ if and only if the flows f and g have the same parity on all edges, that is, for all edges $e \in E(G)$, $f(e)$ is odd if and only if $g(e)$ is odd. An important tool we use to analyse graph structure is the *width*:

Definition 11. We define $\text{width}(G)$ as the minimum number of s - t paths in a DAG G needed to cover all edges in $E(G)$.

Sets of paths and non-negative flows are equivalent, in the sense that you can transform one to the other. Given a set of s - t paths $\mathcal{P}$ on G , we can define a unique flow $f_{\mathcal{P}} = \sum_{p \in \mathcal{P}} p$ that counts the number of paths on every edge. Conversely, given a non-negative flow $f : E(G) \rightarrow \mathbb{N}$, we can define a path cover $\mathcal{P}_f$ by the *Flow Decomposition Theorem*, simply by decomposing the flow into weight 1 paths. It holds that $\text{width}(G) = \min\{\text{Val}(f) \mid f(e) > 0 \forall e \in E(G)\}$ is equal to the value of a minimum covering flow. We say that a path cover induces a flow and vice versa.

Definition 11 is a variant of the more common problem of finding a minimum number of paths to cover all vertices, and both variants can be computed in $O(nm)$ time. This can be done by finding a covering flow that is larger than a minimum covering flow on all edges, and then finding decrementing paths, until there is no s - t path of weight > 1 anymore. This can be formulated as a reduction from minimum flow to a maximum flow instance [1, 18].

3 Parity fixing algorithm

During the decomposition of a flow, flow is subtracted from edges until they are saturated and can not be used anymore. We mitigate the resulting issues that we have mentioned in Section 1.2 by introducing the flow-width of a flow network. We then reformulate the parity fixing approximation algorithm given by Mumey et al. [16] using the flow-width.

²Commonly in the literature, (pseudo-)flows are also required to be skew-symmetric and to be upper-bounded by some capacity function on the edges. These properties play no role in this article.

Definition 14 ([4]). *The class of width-stable DAGs is defined as all G that satisfy $\text{width}(G|_f) \leq \text{width}(G|_g)$ for all flows $0 \leq f \leq g$.*

Width-stable DAGs have been characterised using funnels [10], which are DAGs that generalise in/out-forests: along any s - t path vertices v first satisfy $\deg^-(v) \leq 1 \leq \deg^+(v)$ and then $\deg^-(v) \geq 1 \geq \deg^+(v)$. We also call a u - v path in G a *central path*, if there is a funnel subgraph F with $u, v \in F$, $\deg^-(u) > 1$ and $\deg^+(v) > 1$ in F .

Lemma 15 ([4], Lemma 13). *Let G be an s - t DAG. The following are equivalent.*

1. G is width-stable,
2. For any flow $f \geq 0$ on G , there exists an s - t path in G_f carrying weight $\text{Val}(f)/\text{width}(G_f)$,
3. G has no funnel subgraph with central path.

Conveniently, the width-stable property seamlessly extends to flow-widths. Indeed, the following property shows the impact that the graph structure has on the possible minimal flows that can be defined.

Lemma 16. *Let G be an s - t DAG.*

1. *If G is width-stable, then $\text{fw}(G, f) = \text{width}(G|_f)$ for all flows $f \geq 0$ on G .*
2. *G is width-stable if and only if $\text{fw}(G, f) \leq \text{fw}(G, h)$ for all flows $0 \leq f \leq h$.*

Proof. 1. Let G be width-stable. We can show the statement w.l.o.g. for flows $f > 0$ (i.e. $G|_f = G$). Let f be a positive minimal flow on G , that is, $\text{Val}(f) = \text{fw}(G, f)$. By Lemma 15 there exists an s - t path in G carrying $\text{Val}(f)/\text{width}(G)$. Because f is minimal, the largest weight any s - t path can carry is 1. And thus, $\text{Val}(f)/\text{width}(G) \leq 1$ and $\text{Val}(f) \leq \text{width}(G)$. Since f is positive and thus covers the graph, we also have $\text{Val}(f) \geq \text{width}(G)$. It follows that $\text{fw}(G, f) = \text{width}(G)$. Let f now be a positive, not necessarily minimal covering flow on G . Let h be a minimal flow on G with $h \leq f$. Then $\text{fw}(G, f) \leq \text{fw}(G, h) = \text{width}(G)$. Since also $\text{fw}(G, f) \geq \text{width}(G)$, we have $\text{fw}(G, f) = \text{width}(G)$.

2. If G is width-stable, then the statement follows from Statement 1. Assume G is not width-stable, and let F be a funnel with central path P . We define h and f in the following way: both flows are minimal, with flow 1 on each of the minimal cut-set edges of the funnel. The flow h routes one unit of flow along the central path, while f does not, i.e., $\text{fw}(G, h) = \text{Val}(h) = \text{width}(F) - 1$ and $\text{fw}(G, f) = \text{Val}(f) = \text{width}(F)$. Moreover, we multiply h by $\text{width}(F)$, which does not change $\text{fw}(G, h)$. Since $G|_f \subseteq G|_h$ and $\|f\| \leq \text{width}(F)$, we have $f \leq h$ and $\text{fw}(G, f) > \text{fw}(G, h)$. $\square$

3.2 Parity fixing with minimal flows

We now present the heuristic by Mumey et al. [16] in order to theoretically analyse its performance. The algorithm follows a parity fixing approach, by decomposing the flow f into $\log \|f\|$ flows f_i , so that $f - \sum_{j=0}^{i-1} f_j$ is divisible by 2^i .

The precise description is as follows. Given a flow $f \geq 0$, we want to find a flow g such that $f - g \equiv_2 0$. Mumey et al. [16] have shown that this can be done by finding a min flow, using the following constraints: as lower-bounds 0 on edges where f is even and 1 where f is odd, and as upper-bounds we use f . In other words, they have shown that the following program solves the problem for a given flow network (G, f) :

$$\begin{aligned}
& \min \text{Val}(g), \text{ subject to} \\
& g \text{ is a flow on } G, \\
& 0 \leq g \leq f, \\
& g(e) > 0 \text{ for all } e \in E(G) \text{ where } f(e) \text{ is odd.}
\end{aligned} \tag{3}$$

In the i -th iteration, starting with $i = 0$, we solve Problem 3 on (G, f) with solution f_i , subtract f_i from f and decompose f_i using weight 1 paths, that act as 2^i weighted paths in the decomposition. As a result, f becomes even and we divide it by 2. We then follow up with iteration $i + 1$ and repeat this procedure until $\text{Val}(f) = 0$.

Lemma 17. *Let g be the solution to Problem 3 for a given flow network (G, f) . We have $\text{Val}(g) \leq \text{fw}(G, f)$.*

Proof. Clearly, any flow satisfying the Covering and Upper-bounds constraints from Definition 1 is a feasible solution to Problem 3. $\square$

Theorem 2. *Given an $\text{MFD}_{\mathbb{N}}$ input (G, f) , we can decompose the flow into $\lfloor \log \|f\| \rfloor + 1$ flows f_i , such that*

$$f = \sum_{i=0}^{\lfloor \log \|f\| \rfloor} 2^i f_i \quad \text{and} \quad \text{mfd}_{\{1\}}(G, f_i) = \text{Val}(f_i) \leq \text{fw}(G, f^{(i)}), \quad (2)$$

where $f^{(0)} = f$ and $f^{(i)} = f - \sum_{j=0}^{i-1} 2^j f_j$. This can be done in time $O(nm \log \|f\|)$.

Proof. The algorithm above takes at most $\lfloor \log \|f\| \rfloor + 1$ iterations until the flow is decomposed. By Lemma 17 we have that $\text{Val}(f_i) \leq \text{fw}(G, f^{(i)})$, since $f^{(i)}$ is the flow obtained by the algorithm after i iterations, and the f_i are minimal flows (i.e., flows whose s - t paths all carry at most weight 1) as solutions of Problem 3.

The decomposition into minimal flows can be found in $O(nm \log \|f\|)$ time: Problem 3 can be solved in time $O(nm)$, by reducing the min flow instance to a max flow instance [18]. We solve Problem 3 at most $\lfloor \log \|f\| \rfloor + 1$ times. $\square$

Using this approach, we can decompose f by decomposing each f_i using weight 1 paths. See Algorithm 1 for a pseudo-code description.

Algorithm 1 Approximating $\text{MFD}_{\mathbb{N}}$ [16]

Require: MFD instance (G, f)

Ensure: MFD solution $\mathcal{P}$

```

1:  $i \leftarrow 0$ 
2:  $\mathcal{P} \leftarrow \{\}$ 
3: while  $f > 0$  do
4:    $h \leftarrow$  Optimal solution of Problem 3
5:    $\{P_{i,1}, \dots, P_{i,\text{Val}(h)}\} \leftarrow \text{FD}(G, h)$  Flow decomposition to weight 1 paths
6:    $\{w_{i,1}, \dots, w_{i,\text{Val}(h)}\} \leftarrow \{2^i, \dots, 2^i\}$ 
7:    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(P_{i,j}, w_{i,j})\}$  for  $j = 1, \dots, \text{Val}(h)$ 
8:    $f \leftarrow (f - h)/2$ 
9:    $i \leftarrow i + 1$ 
10: end while

```

We now show our first algorithmic result, which improves a previous approximation ratio for $\text{MFD}_{\mathbb{N}}$ of $O(\log \text{Val}(f))$ by Cáceres et al. [4] on width-stable graphs to $O(\log \|f\|)$. This improves the ratio upper-bound in graphs of large minimum sized cut-sets, as we can have $f(e) = \|f\|$ for all edges $e \in C$ of a minimum cut-set C , which means that $\text{Val}(f) = |C| \cdot \|f\|$.

Theorem 3. *On width-stable DAGs we can approximate $\text{MFD}_{\mathbb{N}}$ given the input (G, f) with ratio $\lfloor \log \|f\| \rfloor + 1$ in runtime $O(n \log \|f\| \cdot (\text{width}(G) + m))$.*

Proof. By Theorem 2 we can express a flow f as the sum of $\lfloor \log \|f\| \rfloor + 1$ flows f_i with $\text{Val}(f_i) \leq \text{fw}(G, f)$. Since G is width-stable, and any flow decomposition is a path cover, we have $\text{fw}(G, f^{(i)}) \leq \text{width}(G|_{f^{(i)}}) \leq \text{width}(G|_f) \leq \text{mfd}_{\mathbb{N}}(G, f)$. We can thus decompose all $\lfloor \log \|f\| \rfloor + 1$ flows f_i with at most $\text{mfd}_{\mathbb{N}}(G, f)$ paths, which gives the approximation ratio.

After expressing the sum in runtime $O(nm \log \|f\|)$ by Theorem 2, we decompose each flow f_i with weight 1 paths, which takes time $O(\text{Val}(f_i) \cdot n) \leq O(\text{width}(G) \cdot n)$, since in DAGs every path is at most of length $n - 1$. We must decompose $\log \|f\|$ flows f_i and obtain the time complexity $O(n \log \|f\| \cdot (\text{width}(G) + m))$. $\square$

4 Parameterised approximation algorithm for $\text{MFD}_{\mathbb{N}}$

We now present a generalisation of Theorem 3 to all DAGs using the parallel-width of an s - t DAG $\text{par-width}(G)$ as graph parameter. It was recently introduced in the work by Deligkas and Meir [6] as the largest minimal cut-set of G , where it was used as a parameter in a new notion of directed graph minors. We show that, despite being used in a different context, there exists a close connection of this definition of directed graph minors to the decomposition of flows, in the sense that contractions of flow-subgraphs are an equivalent definition. We then show that the flow-width generalises the $\text{width}(G)$ and the $\text{par-width}(G)$, which gives us an approximation algorithm based on the two parameters.

4.1 Flows and directed graph minors

Flow decompositions $(\mathcal{P}, w)$ of (G, f) of size k naturally construct a sequence of subgraphs

$$G|_f \supseteq G|_{f-w_1P_1} \supseteq \cdots \supseteq G|_{f-w_1P_1-\cdots-w_{k-1}P_{k-1}} \supseteq G|_0 = \emptyset,$$

for $P_i \in \mathcal{P}$ and $w = (w_1, \dots, w_k) \in \mathbb{N}^k$.

Moreover, flow networks admit natural edge contractions. As Kloster et al. [15] have observed, due to the flow conservation, contracting an edge (u, v) with $\deg^-(v) = 1$ or $\deg^+(u) = 1$ yields a new flow network whose flow decomposition uniquely recovers the corresponding flow decomposition of the original graph. These contractions are sometimes called “Y-to-V”³ and are commonly used to simplify inputs [15, 12].

Definition 18 ([6], Definition 3). *A digraph G' is a directed minor (or d-minor) of a digraph G if G' can be obtained from G by a sequence of the following operations:*

1. **Deletion.** *Deleting an edge (a, b) where $\deg^+(a) > 1$ and $\deg^-(b) > 1$.*
2. **Backward contraction.** *Contracting an edge (a, b) where $\deg^-(b) = 1$.*
3. **Forward contraction.** *Contracting an edge (a, b) where $\deg^+(a) = 1$.*

Note that Backward and Forward contractions are Y-to-V contractions.

Lemma 19. *A DAG G' is a d-minor of a DAG G if and only if G' is a Y-to-V contracted graph of a flow-subgraph of G .*

Proof. First assume that G' is a Y-to-V contracted DAG of a flow-subgraph of G . Let H be the DAG with $H = G|_f$ for some flow $f \geq 0$ on G , such that G' is a Y-to-V contracted graph of H . To construct H from G using d-minor operations, we can alternate between the deletion operation and the contraction operations to delete/contract all edges with $f(e) = 0$, as we can always use at least one operation. To construct G' from H , it is left to do contraction operations, which can be done since G' can be obtained from H with Y-to-V contractions.

Next, assume that G' is a d-minor of G . The edges in G' are contractions of edges in G . Undoing these contractions, we obtain a graph $H \subseteq G$, which we can cover using s - t paths, since the deletion operation enforces the vertices to remain connected to s and t . The paths induce a flow which is positive on $E(H)$ and zero on $E(G - H)$. In addition, G' is a Y-to-V contraction of H . $\square$

Thus, d-minors are compact representations of the flow networks on the subgraphs that can appear during the process of a flow decomposition. This motivates the analysis of hereditary DAG structure.

As a first implication, we show that the class of width-stable DAGs can be described using a forbidden minor.

Definition 20. *We define the graph Ch_k to consist of 4 vertices s, u, v, t and of the following edges: k parallel edges (s, u) , k parallel edges (v, t) , and the three edges $(s, v), (u, v), (u, t)$.*

Lemma 21. *Let G be an s - t DAG. G is width-stable if and only if G is Ch_2 -d-minor free.*

³The name Y-to-V contraction originates from the drawing of the corresponding digraphs. The descender of the letter Y corresponds to the contracted edge.

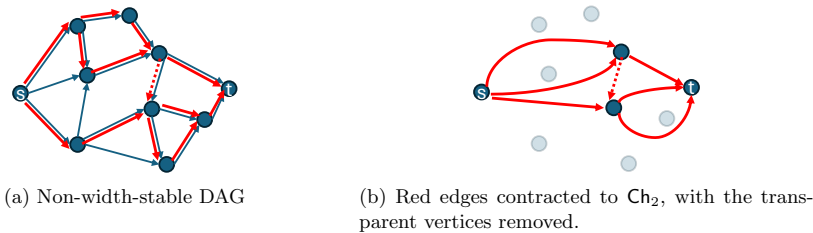


Figure 2: An s - t DAG that is not width-stable. The flow-subgraph in red can be Y-to-V contracted to Ch_2 .

Proof. If G contains Ch_2 as d-minor, then it contains a flow-subgraph H such that Ch_2 is a Y-to-V contracted graph of H . H is then exactly a funnel (of maximum minimal cut-set size 4) with a central path.

If G is not width-stable, it contains a funnel F with central path P from u to v , where $\deg^-(u) \geq 2$ and $\deg^+(v) \geq 2$ with respect to F . This means that there are at least two distinct paths from s to u and two distinct paths from v to t . Since F is a funnel, there also exists a path from u to t avoiding v , and a path from s to v avoiding u . This subgraph of paths has a minimum path cover of size 3, and it induces a flow f on G , such that Ch_2 is a Y-to-V contracted graph of $G|_f$. $\square$

An example of Lemma 21 is illustrated in Figure 2. The lemma gives a natural proof showing that width-stable DAGs generalise series-parallel DAGs, as series-parallel are precisely Ch_1 -d-minor free DAGs [14].

Detecting minors of constant size can be done in polynomial time, which has been shown using several previous results in graph minor theory [6]. However, there is a simple polynomial algorithm that detects if Ch_2 is present in a DAG G as d-minor, which works by computing reachability questions on G .

Corollary 22. *There exists a polynomial time algorithm that detects whether an s - t DAG G is width-stable.*

Proof. First, we compute the number of s - v paths $d_s(v)$ and the number of v - t paths $d_t(v)$ for every $v \in V(G)$. Next, we iterate over all pairs of two internal vertices $u < v$ for a topological order $<$ of G , for which $d_s(u) \geq 2$ and $d_t(v) \geq 2$. This ensures that there is a minor with two parallel edges from s to u and two parallel edges from v to t . With a graph search from u , we can check whether there exists a path to v . Finally, we must check if there exists a path from s to v that avoids u . We can do so by removing u and all its edges from G and by doing a graph search from s in that subgraph. Similarly, we can check if there exists a path from u to t that avoids v . If all these paths exist, we have shown that there is a Ch_2 f-minor with internal vertices u and v . $\square$

4.2 $\text{MFD}_{\mathbb{N}}$ Approximation parameterised by parallel-width

Since the width of a flow-subgraph of G can be larger than the width of G , we use structural parameters to describe the class of DAGs whose flow-subgraphs' widths stay below a given threshold. The work [6] has introduced the *parallel-width* of a DAG G in the context of d-minors as the largest minimal cut-set of G . Let $G_{P(c)}$ be the DAG consisting of two nodes s and t and c parallel edges (s, t) . They have shown that DAGs G with $\text{par-width}(G) < c$ for a constant $c \in \mathbb{N}$ are the class of DAGs with forbidden d-minor $G_{P(c)}$. The following lemma shows why it is relevant for the parity fixing algorithm.

Lemma 4. *For all s - t DAGs G , we have*

$$\text{width}(G) = \min\{\text{fw}(G, f) \mid f > 0\}$$

and

$$\text{par-width}(G) = \max\{\text{fw}(G, f) \mid f \geq 0\}.$$

Proof.

1. $\text{width}(G) \leq \min\{\text{fw}(G, f) \mid f > 0\}$: $\text{fw}(G, f)$ is the value of a positive flow, whereas the $\text{width}(G)$ is the minimum value of a positive flow.
2. $\text{width}(G) \geq \min\{\text{fw}(G, f) \mid f > 0\}$: Consider f to be the induced flow of a minimum path edge cover of G . Then $\text{fw}(G, f) = \text{Val}(f) = \text{width}(G)$.
3. $\text{par-width}(G) \leq \max\{\text{fw}(G, f) \mid f \geq 0\}$: Let $C = \{(u_1, v_1), \dots, (u_\ell, v_\ell)\}$ be the largest minimal cut-set. It was shown in [6] that there exists an out-tree from s , with leaves u_i for $i \in [\ell]$ and an in-tree from the leaves v_i for $i \in [\ell]$ to the root t . The width of this subgraph is $\text{par-width}(G)$, and we can choose for f the induced flow of the minimum path cover of this funnel.
4. $\text{par-width}(G) \geq \max\{\text{fw}(G, f) \mid f \geq 0\}$: The right hand side is the maximum value of all non-negative minimal flows f . When expressing such a minimal flow as its induced path cover, by definition of minimal flows, there exists a minimal cut-set C in $G|_f$ such that every edge in C is covered at most once. Every such cut-set size $|C|$ is upper-bounded by the $\text{par-width}(G)$. Moreover, $|C| \geq \text{Val}(f)$.

□

Theorem 5. *Given a flow network (G, f) with $f > 0$, we can approximate $\text{MFD}_{\mathbb{N}}$ with a ratio of $\left(\frac{\text{par-width}(G)}{\text{width}(G)} \cdot (\log \|f\| + 1)\right)$ in runtime $O(n \log \|f\| \cdot (\text{par-width}(G) + m))$. Moreover, for flow networks with $\text{mfd}_{\mathbb{N}}(G, f) \geq \text{par-width}(G)$, the ratio is $\log \|f\| + 1$.*

Proof. By Lemma 4, we have $\text{fw}(G, f) \leq \text{par-width}(G)$ for any flow $f \geq 0$. Hence, Algorithm 1 returns at most $\text{par-width}(G) \cdot (\log \|f\| + 1)$ many paths. Since $\text{width}(G) = \text{width}(G|_f) \leq \text{mfd}_{\mathbb{N}}(G, f)$, the approximation ratio is $\frac{\text{par-width}(G)}{\text{mfd}_{\mathbb{N}}(G, f)} \cdot (\log \|f\| + 1) \leq \frac{\text{par-width}(G)}{\text{width}(G)} \cdot (\log \|f\| + 1)$. If $\text{mfd}_{\mathbb{N}}(G, f) \geq \text{par-width}(G)$, we moreover obtain a ratio of $\log \|f\| + 1$.

For the runtime, as before, we express f in time $O(nm \log \|f\|)$ as the sum of $\log \|f\| + 1$ flows f_i . We have $\text{Val}(f_i) \leq \text{par-width}(G)$ for all f_i , and thus take $O(n \cdot \text{par-width}(G))$ time to decompose one f_i . □

In practical applications of MFD, it makes sense to consider unary coded flow values, that is, flow networks with input size $O(m \cdot \|f\|)$ (rather than binary coded values with input size $O(m \log \|f\|)$), since the flow captures objects such as information being routed through a network or biological sequence reads.

Corollary 6. *Let $c \in \mathbb{N}$ be a constant. $\text{MFD}_{\mathbb{N}}$ on (G, f) can be solved in quasi-polynomial time $\|f\|^{O(\log \|f\|)} \cdot (m + \log \|f\|)$ when $\text{par-width}(G) \leq c$ and f is coded in unary.*

Proof. Theorem 5 yields an upper-bound for $\text{mfd}_{\mathbb{N}}(G, f)$ of size $\text{par-width}(G) \cdot (\log \|f\| + 1)$. It has previously been shown that $\text{MFD}_{\mathbb{N}}$ is in FPT [15] with parameter $k = \text{mfd}_{\mathbb{N}}(G, f)$, with an implemented tool TOBOGGAN that runs in time $4^{k^2} k^{1.5k} k^{o(k)} 1.765^k \cdot (n + \log \|f\|) = 2^{O(k^2)} \cdot (n + \log \|f\|)$ ([15], Theorem 7). Substituting k for the upper-bound, we obtain a runtime dependent on the $\text{par-width}(G)$ and the logarithm of the largest flow weight in the exponent. If we assume that $\text{par-width}(G)$ is a constant, this running time is $\|f\|^{O(\log \|f\|)} \cdot (n + \log \|f\|)$ by Lemma 4. □

In general, the values $\text{fw}(G, f^{(i)})$ can grow above $\text{mfd}_{\mathbb{N}}(G, f)$. In the following, we show that the approximation ratio of Algorithm 1 can be as large as $\Omega(\|f\|)$ in some classes of graphs, which increases the gap between $\text{MFD}_{\mathbb{N}}$ and $\text{MFD}_{\mathbb{Z}}$. Consider the MFD instance $(G_{(k, \ell)}, f_k)$ defined in Figure 3. The following lemma shows the idea of the construction.

Lemma 23. *For all $c > 1$, there exist $k, \ell > 0$, such that*

$$\frac{\text{par-width}(G_{(k, \ell)})}{\text{mfd}_{\mathbb{N}}(G_{(k, \ell)}, f_k)} > c.$$

Proof. Clearly, $\text{par-width}(G_{(k, \ell)}) = 6k + \ell(k + 2)$. To decompose the flow in an efficient way, we can decompose all the maximum cut-set edges of flow 2 with k paths, which also saturates all the connecting

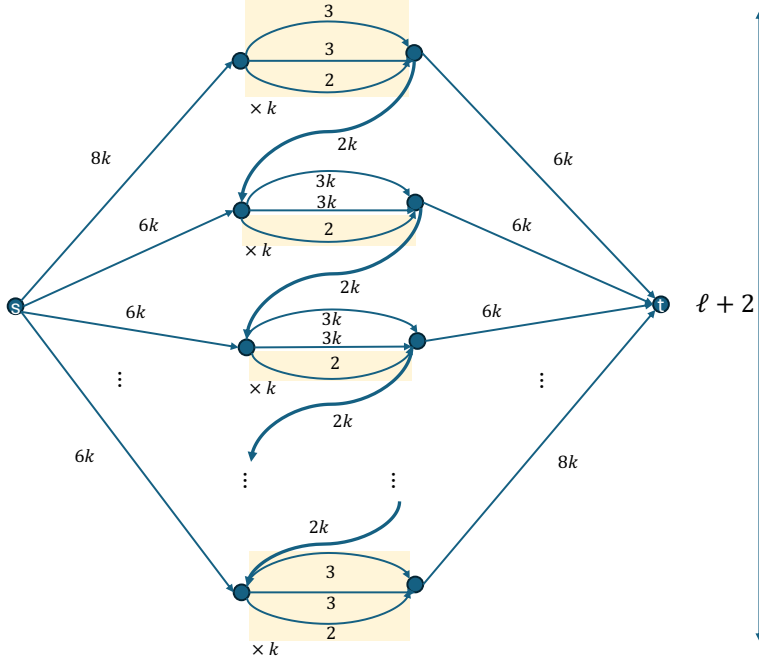


Figure 3: MFD instance $(G_{(k, \ell)}, f_k)$. The yellow highlighted edges are copied k times, and there are $\ell + 2$ total central gadgets (that is, $\deg^+(s) = \ell + 2$). Thus, $\text{par-width}(G_{(k, \ell)}) = 6k + \ell(k + 2)$, and we call these edges the maximum cut-set edges. There are two central gadgets consisting of $3k$ maximum cut-set edges and ℓ central gadgets each consisting of $k + 2$ maximum cut-set edges.

central edges between them (in bold). We then use additional $4k + 2\ell$ paths to fully decompose the graph. In total, $\text{mfd}_{\mathbb{N}}(G_{(k, \ell)}, f_k) \leq 5k + 2\ell$, and thus,

$$\frac{\text{par-width}(G_{(k, \ell)})}{\text{mfd}_{\mathbb{N}}(G_{(k, \ell)}, f_k)} \geq \frac{6k + \ell(k + 2)}{5k + 2\ell} \xrightarrow{k \rightarrow \infty} \frac{6 + \ell}{5}.$$

This shows that for $\ell > 5c - 6$, there exists $k > 0$, such that $\text{par-width}(G_{(k, \ell)})/\text{mfd}_{\mathbb{N}}(G_{(k, \ell)}, f_k) > c$. $\square$

The strategy is now to show that the approximation algorithm uses almost $\text{par-width}(G_{(k, \ell)})$ many paths to cover the odd edges in the second iteration. This will imply the approximation factor of $\Omega(\|f\|)$. Let $\mathcal{P}_i$ be the set of paths that the approximation algorithm uses to cover the odd edges in iteration i . That is, $|\mathcal{P}_i|$ is the optimal solution of Problem 3 in iteration i of the algorithm.

Lemma 24. *In the worst case, Algorithm 1 is a factor $\Omega(\|f\|)$ approximation algorithm.*

Proof. Let k be odd. In the first iteration the approximation algorithm fixes the parity of the $4k$ edges whose flow weight is each 3, traversing the 2ℓ edges whose flow weight is each $3k$ (because k is odd), using $2k$ many paths. This causes the paths to decompose the central connecting edges in bold of flow $2k$. In the second iteration ($i = 1$), after dividing the flow by 2, there are $|\mathcal{P}_1| = 6k + k\ell$ many odd edges that are pairwise unreachable. Thus, if we have $k = \ell$,

$$\frac{|\mathcal{P}_1|}{\text{mfd}_{\mathbb{N}}(G_{(k, k)}, f_k)} \geq \frac{6k + k^2}{6k} = \Theta(k).$$

The approximation factor follows, because $k = \frac{1}{8}\|f\| = \Theta(\|f\|)$, and the number of paths returned by the approximation algorithm is at least $|\mathcal{P}_1|$. $\square$

5 MFD hardness results

A previous reduction [24] has shown that width-stable $\text{MFD}_{\mathbb{N}}$ instances of arbitrary width are strongly NP-hard. Here we show that $\text{MFD}_{\mathbb{N}}$ remains NP-hard even when G has a small width by showing that:

1. $\text{MFD}_{\mathbb{N}}$ is *strongly* NP-hard on DAGs G of width 3.
2. $\text{MFD}_{\mathbb{N}}$ is NP-hard on DAGs G of width 2.

Note that $\text{width}(G) = 2$ if and only if $\text{par-width}(G) = 2$, because, since $\text{width}(\text{Ch}_2) = 3$, every DAG of width 2 is width-stable. Moreover, DAGs G with $\text{width}(G) = \text{par-width}(G) = 1$ are s - t paths, which obtain a unique flow decomposition. Our results are thus tight with respect to the width.

Theorem 7. $\text{MFD}_{\mathbb{N}}$ on G is strongly NP-hard even when $\text{width}(G) = 3$.

Proof. We will show a reduction from the 3-PARTITION problem to $\text{MFD}_{\mathbb{N}}$ on G of width 3. Let $a_1, \dots, a_{3q}, B \in \mathbb{N}$ such that $a_i \in (B/4, B/2)$. We want to find a partition of the $a_1, \dots, a_{3q}$ to sets $S_1, \dots, S_q$ such that, for all i , $\sum_{a \in S_i} a = B$. Note that this restriction implies that $\forall i : |S_i| = 3$.

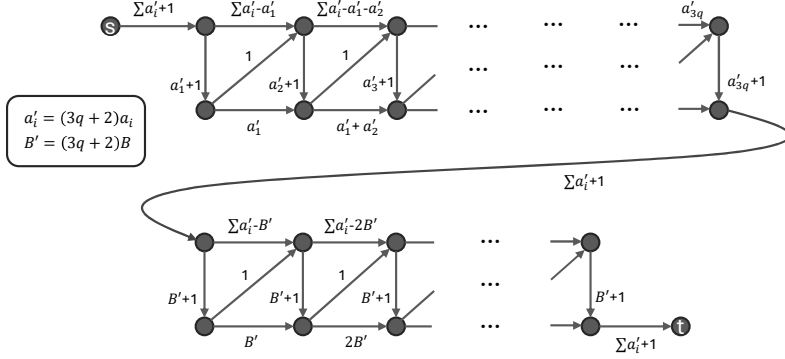


Figure 4: An $\text{MFD}_{\mathbb{N}}$ instance of width 3 for solving 3-PARTITION problem.

Consider the $\text{MFD}_{\mathbb{N}}$ instance G in Figure 4, which we can divide into the top and the bottom articulation components. Each vertical edge in the top component represents an a_i and each vertical edge in the bottom component represents B . We claim that there is a solution to the 3-PARTITION problem if and only if this $\text{MFD}_{\mathbb{N}}$ instance on G has a solution of size $3q + 1$.

First, we show that, if we have a solution to the 3-PARTITION problem, then we can decompose G into $3q + 1$ weighted paths. This can be done by routing one path of weight 1 to saturate all the diagonal edges. Then, for each $a_i \in S_j$, we route a path of weight $(3q + 2)a_i$ through the i^{th} vertical edge in the top component and through the j^{th} vertical edge in the bottom component.

Now, we will show that, if we can decompose G into $3q + 1$ weighted paths, then we can construct a solution to the 3-PARTITION. Let $F = \{(P_i, w_i) \mid i \in [3q + 1]\}$ be the set of weighted paths in the solution in a non-decreasing order of w_i . Let $U \subseteq F$ be the set of paths with a weight of 1 each. Note that all the diagonal edges must be saturated by U . Since $|U| \leq 3q + 1$ and each vertical edge in the top component has a flow of at least $3q + 2$, they are not saturated by U . This means that we need at least $3q$ paths to saturate all vertical edges in the top component, or in other words, $|F \setminus U| \geq 3q$. Hence, $|U| = 1$, and a path of weight 1 routes through all diagonal edges in both the top and bottom components. For the remaining $3q$ paths, since each path can only pass one vertical edge in the top components, it must saturate one vertical edge in the top component. To construct a solution to the 3-PARTITION problem, we put a_i in a set S_j if the path that saturates the i^{th} vertical edge in the top component uses the j^{th} vertical edge in the bottom component.

□

Note that the instance in Figure 4 is not width-stable because after removing the diagonal edges using a single path of weight 1, we obtain the $\text{MFD}_{\mathbb{N}}$ instance of the known reduction of width $3q$.

Theorem 8. $\text{MFD}_{\mathbb{N}}$ on (G, f) is NP-hard when f is coded in binary, even when $\text{width}(G) = 2$.

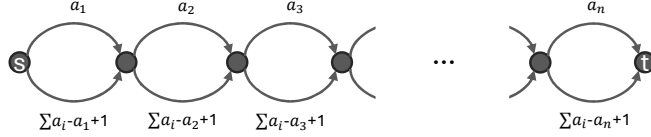


Figure 5: An $\text{MFD}_{\mathbb{N}}$ instance of width 2 for solving GENERATING SET problem. Note that this multi-graph can be turned into a graph by subdividing each edge.

Proof. We will show a reduction from the GENERATING SET problem to $\text{MFD}_{\mathbb{N}}$ on G of width 2. In the GENERATING SET problem, we are given a set of positive integers $A = \{a_1, \dots, a_n\}$ and a positive integer k , and we want to decide if there is a set $Z = z_1, \dots, z_k$ such that every element $a \in A$ is the sum of some subset of the elements in Z . The GENERATING SET problem is known to be NP-hard [5] and the proof of that uses integers that grow exponentially (thus, not proving strong NP-hardness).

Consider the $\text{MFD}_{\mathbb{N}}$ instance in Figure 5. Let $e_i^{(t)}$ and $e_i^{(b)}$ be the i^{th} top and bottom edge from the left, respectively. We construct G of width 2 by using a_i as a weight of $e_i^{(t)}$. We also let the total $(s-t)$ -flow have a value of $\sum_{i=1}^n a_i + 1$. We will show that GENERATING SET problem has a solution of size k if and only if $\text{MFD}_{\mathbb{N}}$ on G has a solution of size $k + 1$.

First, we show that, if we have a solution of GENERATING SET of size k , we can obtain a solution of $\text{MFD}_{\mathbb{N}}$ of size $k + 1$. Let $Z = \{z_1, \dots, z_k\}$ be a solution of GENERATING SET. We have that, for all $i \in [n]$, there is $\chi_{ij} \in \{0, 1\}^{n \times k}$ such that $a_i = \sum_{j=1}^k \chi_{ij} z_j$. Let w.l.o.g. $\sum_{i=1}^n \chi_{ij} \geq 1$ for all $j \in [k]$. In the corresponding $\text{MFD}_{\mathbb{N}}$ solution, for $j \in [k]$, we route a path P_j of weight z_j via $e_i^{(t)}$ when $\chi_{ij} = 1$, and route P_j via $e_i^{(b)}$ when $\chi_{ij} = 0$. After we route $P_1, \dots, P_k$, all the top edges will be saturated. Finally, we route P_{k+1} of remaining weight via all bottom edges.

Next, we show that, if we have a solution of $\text{MFD}_{\mathbb{N}}$ of size $k + 1$, we can obtain a solution of GENERATING SET of size k . Let $\{(P_1, w_1), \dots, (P_{k+1}, w_{k+1})\}$ be the $\text{MFD}_{\mathbb{N}}$ solution. Note that the total flow in this instance has weight $\sum_{i \in [n]} a_i + 1$, and the total of the weight of all P_j that use at least one top edge is at most $\sum_{i \in [n]} a_i$. Since the total flow is strictly more than the total weight of all P_i that use at least one top edge, there must be one path in our solution that uses only bottom edges. W.l.o.g. Let P_{k+1} be such a path. Notice that, for the remaining k paths $P_1, \dots, P_k$, if all their weights are distinct, we claim that $\{w_1, \dots, w_k\}$ is a solution of GENERATING SET. This can be done by setting $\chi_{ij} = 1$ when P_j is routed via $e_i^{(t)}$, and $\chi_{ij} = 0$ when P_j is routed via $e_i^{(b)}$. Now, when their weights are not distinct, we have a multiset that satisfies the GENERATING SET problem. We can turn this multiset into a set by the following. Let w be the smallest positive integer in this multiset and d be the number of copies. We replace these integers with $w, 2w, \dots, dw$ and adjust χ accordingly. We can repeat this process until all integers are distinct. The process is terminated in at most k rounds since we obtain at least one distinct element w after each round. □

6 Conclusions

In this article we examined the performance of an $\text{MFD}_{\mathbb{N}}$ heuristic. We have introduced the flow-width of a flow network, which we have used to relate minimal flows to graph structure. We have shown that expressing the input flow as the sum of minimal flows can lead to efficient approximations for minimum flow decompositions, when the flow-width grows by only a small factor throughout the decomposition. In particular, we have shown that we can decompose any flow $f > 0$ on G in at most

$\text{par-width}(G) \cdot (\lceil \log \|f\| \rceil + 1)$ paths. We obtain an approximation ratio of $O(\lceil \log \|f\| \rceil + 1)$ on the class of inputs with $\text{par-width}(G)/\text{mfd}_{\mathbb{N}}(G, f) < c$ for some constant c .

Moreover, we have answered an open problem regarding the computational complexity parameterised by the width of $\text{MFD}_{\mathbb{N}}$. We have shown that $\text{MFD}_{\mathbb{N}}$ on graphs G with $\text{width}(G) = 2$ or $\text{par-width}(G) \leq c$ obtains a quasi-polynomial runtime when the flow is coded in unary and is NP-hard for binary coded flows. When the width is equal to 3, the problem remains even strongly NP-hard.

We leave open questions on the approximability of $\text{MFD}_{\mathbb{N}}$. Can we efficiently approximate $\text{MFD}_{\mathbb{N}}$ on the remaining class of inputs where $\text{par-width}(G)/\text{mfd}_{\mathbb{N}}(G, f) > c$ for any $c > 1$? Is $\text{MFD}_{\mathbb{N}}$ in APX, or can we show that a constant factor approximation algorithm is unlikely to exist? If so, what about graphs of constant parallel-width?

References

- [1] Ravindra K Ahuja, James B Orlin, and Thomas L Magnanti. *Network flows: theory, algorithms, and applications*. Prentice-Hall, 1993.
- [2] Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *Research in Computational Molecular Biology: 24th Annual International Conference, RECOMB 2020, Padua, Italy, May 10–13, 2020, Proceedings 24*, pages 221–222. Springer, 2020.
- [3] 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.
- [4] Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I Tomescu, and Lucia Williams. Width helps and hinders splitting flows. *ACM Transactions on Algorithms*, 20(2):1–20, 2024.
- [5] Michael J. Collins, David Kempe, Jared Saia, and Maxwell Young. Nonnegative integral subset representations of integer sets. *Information Processing Letters*, 101(3):129–133, 2007. URL: <https://www.sciencedirect.com/science/article/pii/S0020019006002663>, doi:10.1016/j.ipl.2006.08.007.
- [6] Argyrios Deligkas and Reshef Meir. Directed Graph Minors and Serial-Parallel Width. In Igor Potapov, Paul Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*, volume 117 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2018.44>, doi:10.4230/LIPIcs.MFCS.2018.44.
- [7] Fernando H. C. Dias, Lucia Williams, Brendan Mumey, and Alexandru I. Tomescu. Fast, flexible, and exact minimum flow decompositions via ILP. 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 230–245. Springer, 2022. doi:10.1007/978-3-031-04749-7_14.
- [8] Fernando HC Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics*, 39(11):btad640, 2023.
- [9] David Eppstein. Parallel recognition of series-parallel graphs. *Information and Computation*, 98(1):41–55, 1992.
- [10] Marcelo Garlet Millani, Hendrik Molter, Rolf Niedermeier, and Manuel Sorge. Efficient algorithms for measuring the funnel-likeness of dags. *Journal of Combinatorial Optimization*, 39:216–245, 2020.
- [11] Thomas Gatter and Peter F Stadler. Ryūtō: network-flow based transcriptome reconstruction. *BMC bioinformatics*, 20:1–14, 2019.

- [12] Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu. Accelerating ILP Solvers for Minimum Flow Decompositions Through Search Space and Dimensionality Reductions. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms (SEA 2024)*, volume 301 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SEA.2024.14>, doi:10.4230/LIPIcs.SEA.2024.14.
- [13] 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.
- [14] Ron Holzman and Nissan Law-Yone. Network structure and strong equilibrium in route selection games. *Mathematical social sciences*, 46(2):193–205, 2003.
- [15] Kyle Kloster, Philipp Kuinke, Michael P O’Brien, Felix Reidl, Fernando Sánchez Villamil, 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.
- [16] 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.
- [17] Nils Olsen, Natalia Kliwer, and Lena Wolbeck. A study on flow decomposition methods for scheduling of electric buses in public transport based on aggregated time-space network models. *Central European Journal of Operations Research*, pages 1–37, 2022.
- [18] James B Orlin. Max flows in $o(nm)$ time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 765–774, 2013.
- [19] Mihaela Perteau, Geo M Perteau, 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.
- [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 1–10. Springer, 2013.
- [23] Jacobo Valdes, Robert E Tarjan, and Eugene L Lawler. The recognition of series parallel digraphs. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 1–12, 1979.
- [24] Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe 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.
- [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.

Paper IV

Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu

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

In *Proceedings of the 22nd International Symposium on Experimental Algorithms (SEA 2024)*,

Leibniz International Proceedings in Informatics (LIPIcs), 2024, pages 14:1-14:19.

Copyright © Andreas Grigorjew, Fernando H. C. Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I. Tomescu.

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

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 $34\times$ 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 $219\times$ 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 $\rightarrow$ Network flows; Applied computing $\rightarrow$ 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 be used to reconstruct viral quasiespecies [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 $O(\log \text{VAL}(f))$ (where $\text{VAL}(f)$ 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 $34\times$ 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 $219\times$ 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 $P : E \rightarrow \{0, 1\}$ 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 $P = (e_1, \dots, e_{|P|})$ we denote by $P[\ell..r] := (e_\ell, \dots, e_r)$ a subpath of length $r - \ell + 1$. A *flow network* is the tuple $G = (V, E, f)$, where $f : E \rightarrow \mathbb{Z}^+$ 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 $\mathcal{P} = (P_1, \dots, P_k)$ with their associated weights $w = (w_1, \dots, w_k) \in (\mathbb{Z}^+)^k$, such that:

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

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 $\sum_{i \in \{1, \dots, k\}} P_i(u, v) w_i \in [L_{uv}, R_{uv}], \forall (u, v) \in E$. 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.

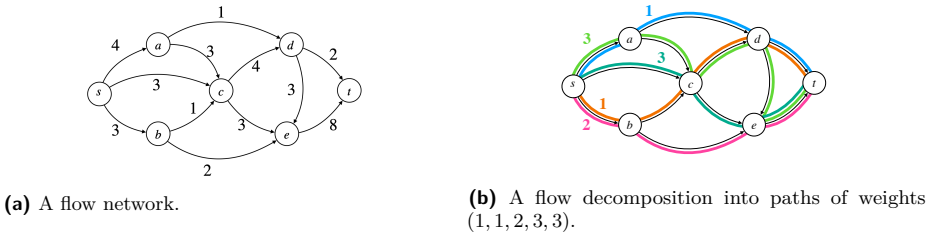


Figure 1 Example of a flow network in (1a) and a decomposition of it into five s - t paths in (1b).

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 , $i \in \{1, \dots, k\}$, 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 $i \in \{1, \dots, k\}$:

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

Constraint (1) is then modeled as:

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

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 $\sum_{i \in \{1, \dots, k\}} x_{uvi} w_i$ 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 $i \in \{1, \dots, k\}$, 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:

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

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

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 $P = (e_1, \dots, e_{|P|})$ with $e_i = (u_i, u_{i+1})$, define its excess flow:*

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

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 $P = (e_1, \dots, e_{|P|})$ 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 $r = 2, \dots, |P|$, 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 decompositions 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.

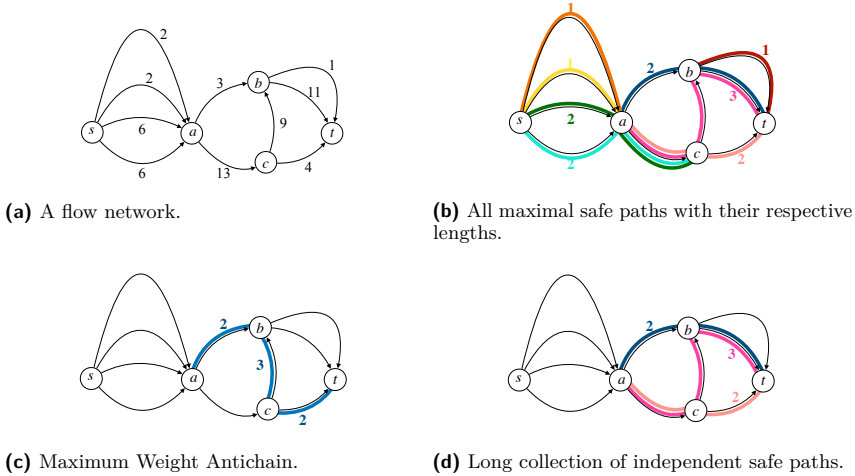


Figure 2 Application of safe paths in introducing new constraints into previous ILP models. A flow network is displayed in Figure 2a. From this network, all maximal safe paths (that cannot be extended left or right) can be calculated (displayed in different colours with their respective lengths in Figure 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 $\mathcal{P}(Q) = \{P_1, \dots, P_{|Q|}\}$ 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 $\mathcal{P}(Q)$:

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

Additionally, in order to further optimize the choice of the antichain Q , we consider the following notion. Given a weight function $a : E \rightarrow \mathbb{N}$, 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: $w_{i+1} \leq w_i, \forall i \in \{|Q| + 1, \dots, k - 1\}$. 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 $P = (e_1, \dots, e_{|P|})$ with $e_i = (u_i, u_{i+1})$ define its inexact excess flow as*

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

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 $w \in \mathbb{Z}^{+|P|}$ routed through edge e_1 passes through all of P . This means they all leave P through edges $\text{OUT} := \{(u_i, v) \in E \mid i \in \{2, \dots, k\}, v \neq u_{i+1}\}$ 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

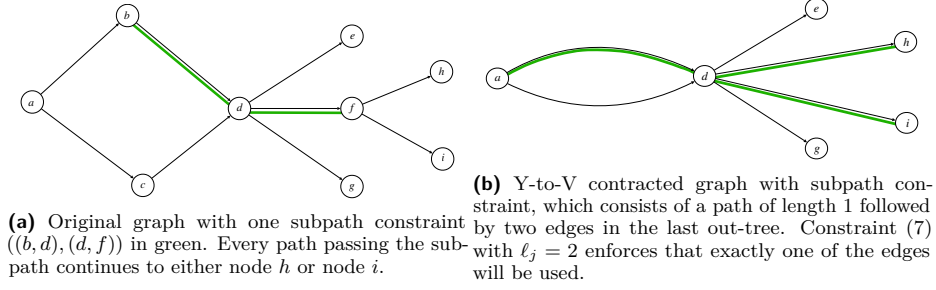
Koster et al. [11, Lemma 4.1] used optimization to reduce the graph size in a pre-processing step by suitably contracting all edges entering nodes with in-degree 1, and respectively, exiting from nodes with out-degree 1, as we review below. This reduction has sometimes been used, under the name “Y-to-V”, 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 extend 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 $w_i \in V$, $i = 1, \dots, \ell$ 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 $f' : E' \rightarrow \mathbb{N}$ with $f'(u, w_i) = f(v, w_i)$ for $i = 1, \dots, \ell$. 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 $T_1, \dots, T_{\ell_j}$, 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 $T_1, \dots, T_{\ell_j-1}$, 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:



■ **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).

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

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

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 $T_1, \dots, T_{\ell_j-1}$ 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 unique 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

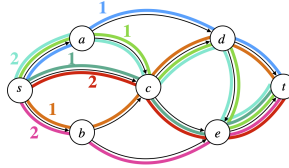
of the single in-edge of u (respectively for the sum of all outgoing edges and the interval $[L_{\text{out}}, R_{\text{out}}]$ of the single out-edge of u). Note that the second inequality is always fulfilled since $L_{uv} \leq R_{uv}$. The Y-to-V reduction for MIFD works the same as the Y-to-V reduction for MFD, but defines intervals $[L'_{uw_i}, R'_{uw_i}]$ (resp. $[L'_{w_i u}, R'_{w_i u}]$) 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 $W = \{w_1, \dots, w_\ell\}$, 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 $W = \{w_1, \dots, w_\ell\} \subseteq \mathbb{Z}^+$, find a minimum-sized set of s - t paths $\mathcal{P} = (P_1, \dots, P_k)$ with associated weights $(\tilde{w}_1, \dots, \tilde{w}_k)$, with each $\tilde{w}_i \in W$, such that $\sum_{i \in \{1, \dots, k\}} P_i(u, v) \tilde{w}_i = f_{uv}$, $\forall (u, v) \in E$.

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:

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

Subject to:

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

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

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

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 $W = \{1, \dots, c\}$ solves MFD on instances where the flows are upper bounded by a constant c by Lemma 8. $\blacktriangleleft$

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

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

Proof. Let $\mathcal{P} = \{P_1, \dots, P_k\}$ with associated weights $\{w_1, \dots, w_k\}$ 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, $|\mathcal{P}_w| \leq k \lceil \log \|f\| \rceil + 1$.

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 $w = \sum_{j=0, \dots, \lceil \log \|f\| \rceil} b_j 2^j$ for $b_j \in \{0, 1\}$, then $w_i = \sum_{j=0, \dots, \lceil \log \|f\|/i \rceil} 2^{ij} (b_{ij} + 2b_{ij+1} + \dots + 2^{i-1}b_{ij+i-1}) = 2^0(b_0 + 2b_1 + \dots + 2^{i-1}b_{i-1}) + 2^i(b_i + 2b_{i+1} + \dots + 2^{i-1}b_{2i-1}) + \dots$. The sums $(b_{ij} + 2b_{ij+1} + \dots + 2^{i-1}b_{ij+i-1}) \leq 2^i - 1$ denote how often we need to copy P using weight 2^{ij} . $\blacktriangleleft$

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 $\Delta(|\mathcal{P}|)$ 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 $27\text{--}31\times$ faster than MFD-original on average and for Catfish Mouse ($\min k \geq 21$), MFD-optimized is $23\times$ 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.

	min k	Catfish heuristic			MFDW heuristic			MFD-original			MFD-optimized		
		Avg.	Total	$\Delta(\mathcal{P})$	Avg.	Total	$\Delta(\mathcal{P})$	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 $11\times$, $69\times$ and $121\times$ 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.

	min k	CoasterHeuristic			MFDSC-original			MFDSC-optimized		
		Avg.	Total	$\Delta(\mathcal{P})$	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 $30\times$ faster on SRR020730 Salmon, up to $219\times$ faster on Catfish

Human and up to $100\times$ 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.

	min k	IFDSolver heuristic			MIFD-original			MIFD-optimized		
		Avg.	Total	$\Delta(P)$	Avg.	Total	#Solved	Avg.	Total	#Solved
SRK020730 Shannon	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 $34\times$ 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. Ryütō: 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 Kinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villamil, 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

$$\forall i \in \{1, \dots, k\} : \quad \sum_{(s,v) \in E} x_{svi} = 1 \quad (10a)$$

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

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

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

$$\forall (u, v) \in E, \forall i \in \{1, \dots, k\} : \quad \phi_{uvi} \leq f_{\max} x_{uvi} \quad (10e)$$

$$\phi_{uvi} \leq w_i \quad (10f)$$

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

$$w_i \in \mathbb{Z}^+ \quad \forall i \in \{1, \dots, k\}, \quad (10h)$$

$$x_{uvi} \in \{0, 1\} \quad \forall (u, v) \in E, \forall i \in \{1, \dots, k\}, \quad (10i)$$

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

A.2 MFD-optimized

Equations (10) together with:

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

$$w_{i+1} \leq w_i, \quad \forall i \in \{|Q| + 1, \dots, k - 1\} \quad (11b)$$

A.3 MFDW

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

Subject to:

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

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

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

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 $O(|E(\mathcal{D})| \cdot \sum_{S \in \mathcal{S}} |S|) \subseteq O(|\mathcal{S}|^2 \cdot \sum_{S \in \mathcal{S}} |S|)$. ◀

TIETOJENKÄSITTELYTIETEEN OSASTO
PL 68 (Pietari Kalmin katu 5)
00014 Helsingin yliopisto

DEPARTMENT OF COMPUTER SCIENCE
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki, FINLAND

JULKAISUSARJA A

SERIES OF PUBLICATIONS A

Reports are available on the e-thesis site of the University of Helsinki.

- A-2023-1 E. Zosa: Analysis of News Media with Topic Models. 79+67 pp. (Ph.D. Thesis)
- A-2023-2 R. Walve: Improving Contiguity and Accuracy in Genome Assembly. 40+53 pp. (Ph.D. Thesis)
- A-2023-3 L. Leppänen: Methods for Automated Generation of Natural-Language Reports. 72+ 73 pp. (Ph.D. Thesis)
- A-2023-4 S. Mäkinen: Increasing Release Frequency by Accelerating Software Development Cycles in Software Engineering. 130+66 pp. (Ph.D. Thesis)
- A-2023-5 C. Rajani: From Optics to Robotics: Machine Learning in the Real World. 66+91 pp. (Ph.D. Thesis)
- A-2023-6 J. Lagus: Transformations and document similarities in word embedding spaces. 84+ 60 pp. (Ph.D. Thesis)
- A-2023-7 M. Luotamo: Advances in Region-Based Multisource Machine Learning for Remote Sensing. 86+80 pp. (Ph.D. Thesis)
- A-2023-8 R. Beigaitė: Machine Learning Methods for Globally Structured Multi-Target Data. 42+54 pp. (Ph.D. Thesis)
- A-2023-9 T. Lehtonen: Computational Approaches to Reasoning in Structured Argumentation. 94+117 pp. (Ph.D. Thesis)
- A-2023-10 E. Soisalon-Soininen: Neural Transfer Learning for Truly Low-Resource Natural Language Processing. 121 pp. (Ph.D. Thesis)
- A-2023-11 L. Osmani: Multicloud Management Plane: Experiments with Scientific and Telecom Clouds. 73+43 pp. (Ph.D. Thesis)
- A-2023-12 L. Myllyaho: On Quality Assurance of Intelligent Systems. 63+81 pp. (Ph.D. Thesis)
- A-2023-13 A. Merchant: Applications of Node Embeddings to Learning and Mining on Graphs. 67+74 pp. (Ph.D. Thesis)
- A-2023-14 S. Schmidt: Unitigs Are Not Enough: the Advantages of Superunitig-Based Algorithms in Bioinformatics. 79+104 pp. (Ph.D. Thesis)
- A-2023-15 O. Nyberg: Uplift Modeling with Click-Stream Data. 72+79 pp. (Ph.D. Thesis)
- A-2023-16 M. Cáceres: Parametrised and Safe & Complete Graph Algorithms for Bioinformatics. 68+72 pp. (Ph.D. Thesis)
- A-2023-17 V. Hyvönen: A Machine Learning Approach to Approximate Nearest Neighbor Search. 88+107 pp. (Ph.D. Thesis)
- A-2023-18 A. Zuñiga Corrales: Pervasive Data Science: From Data Collection to End-User Applications. 72+132 pp. (Ph.D. Thesis)
- A-2023-19 Q. Guo: Developing Clinical Potential of Mutational Signatures: Insights from Formalin-Fixed Biopsies and MSI-High Tumours. 88+109 pp. (Ph.D. Thesis)
- A-2024-1 A. Meijer-van de Griend: Advances in Quantum Compilation in the NISQ Era. 72+ 106 pp. (Ph.D. Thesis)
- A-2024-2 I. Ahmed: Haptic interaction with affective agents in virtual reality. 86+70 pp. (Ph.D. Thesis)
- A-2024-3 B. Zhukova: New space-time trade-offs for pattern matching with compressed indexes. 110+62 pp. (Ph.D. Thesis)

- A-2024-4 S. Ivanova: Language Technology Tools for Low-Resource Languages - Five Cases for Sakha, Norwegian, and Finnish. 72+55 pp. (Ph.D. Thesis)
- A-2024-5 G. Kia: Utilization of Machine Learning Algorithms and Radio Frequency Signals to Remove the Barriers of Low-Cost Accurate Seamless Localization. 58+82 pp. (Ph.D. Thesis)
- A-2024-6 J. Aldahdooh: Building a comprehensive drug-target knowledge base using biomedical text mining. 105+57 pp. (Ph.D. Thesis)
- A-2024-7 M. Leinonen: Optical Maps in Genome Assembly and Long k -mer Extraction. 57+95 pp. (Ph.D. Thesis)
- A-2024-8 A. Björklund: Interpretable and explainable machine learning for natural sciences. 64+119 pp. (Ph.D. Thesis)
- A-2024-9 H. Hälvä: Nonlinear Independent Component Analysis with Structured Priors. 62+ 94 pp. (Ph.D. Thesis)
- A-2024-10 J. Harviainen: Advances in Sampling and Counting Bipartite Matchings and Directed Acyclic Graphs. 63+62 pp. (Ph.D. Thesis)
- A-2024-11 R. Savvides: Statistical methods for testing visual patterns, selecting models, and bounding model errors. 64+83 pp. (Ph.D. Thesis)
- A-2024-12 G. Akman: Privacy and Mobility Aspects in Authentication and Key Agreement Protocols. 130+128 pp. (Ph.D. Thesis)
- A-2024-13 A. Mahadevan: Scaling and Maintaining Machine Learning Pipelines. 68+96 pp. (Ph.D. Thesis)
- A-2025-1 Z. Hussain: Learning Command Syntax and Detecting Similarities for Enhanced Cybersecurity through Data Analysis. 71+68 pp. (Ph.D. Thesis)