Width Helps and Hinders Splitting Flows

Manuel Cáceres  
Department of Computer Science,  
University of Helsinki, Finland

Massimo Cairo  
Department of Computer Science,  
University of Helsinki, Finland

Andreas Grigorjew  
Department of Computer Science,  
University of Helsinki, Finland

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

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

Romeo Rizzi  
Department of Computer Science,  
University of Verona, Italy

Alexandru I. Tomescu  
Department of Computer Science,  
University of Helsinki, Finland

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 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 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 an $O(\log |X|)$-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 → Network flows

Keywords and phrases  
Flow decomposition, approximation algorithms, graph width

Digital Object Identifier  
10.4230/LIPIcs.ESA.2022.31

Related Version  

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), 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...
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 \( \lambda \) 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 MFD\(_{\mathbb{N}}\) and MFD\(_{\mathbb{Z}}\) throughout the paper. While 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 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 MFD\(_{\mathbb{Z}}\).
In Section 4 we consider connections between the width and a popular heuristic algorithm for MFD$_N$ which we call greedy-weight\textsuperscript{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 MFD$_N$ 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 MFD$_N$, 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 MFD$_N$. In this paper, we disprove the conjecture of [14], further corroborating the gap between MFD$_N$ and MFD$_Z$.

## 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

\textsuperscript{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.
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.

\begin{itemize}
  \item \textbf{Definition 1.} Given a flow \(X\), a flow decomposition of \((G, X)\) of size \(k\) is a family of \(s\)-\(t\) paths \(P = (P_1, \ldots, P_k)\) with weights \((w_1, \ldots, w_k) \in \mathbb{Y}^k\) such that \(X = w_1P_1 + \cdots + w_kP_k\).
  \item \textbf{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 \(\mathbb{Y}\).
\end{itemize}

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 \textit{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 if \(Y(u, v)\) is odd.

\begin{itemize}
  \item \textbf{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 \(\text{width}(G)\).
\end{itemize}

\begin{figure}[h]
\centering
\begin{subfigure}{0.4\textwidth}
  \centering
  \includegraphics[width=\textwidth]{positive_weights}
  \caption{If negative weights are allowed, the four paths decompose the flow with weights 4, 5, 8, and \(-3\) (dark blue).}
\end{subfigure}
\begin{subfigure}{0.4\textwidth}
  \centering
  \includegraphics[width=\textwidth]{negative_weights}
  \caption{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.}
\end{subfigure}
\caption{A positive flow admitting a decomposition into four paths only if negative weights are allowed.}
\end{figure}
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 \(S\).

\begin{lemma}[1, 17]\end{lemma}

Let \(G = (V, E)\) be a DAG, and \(S \subseteq E\). A flow \(C : E \to \mathbb{N}\) 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 idea behind our approximation algorithm for MFD is that a flow \(X : E \to \mathbb{Z}\) 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 \(X\).

\begin{lemma}\end{lemma}

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

\begin{proof}\end{proof}

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

Notice that every node of \(G_{\text{odd}}\), except possibly \(s\) and \(t\), has even degree, due to conservation of flow. Thus, \(G_{\text{odd}}\) 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 \(E^{+}_{\text{odd}}\) 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]).

\begin{corollary}\end{corollary}

Any (non-zero) flow \(X : E \to \mathbb{Z}\) can be written as \(X = \sum_{i=0}^{\lceil \log \|X\| \rceil} 2^i \cdot Y_i\), where \(Y_i : E \to \mathbb{Z}\) is a flow with \(\|Y_i\| \leq 1\) for all \(i\).
(a) Unitary flow $X$ on a graph $G$ and a decomposition of it into four paths, two of weight 1 in orange (see (e)), and two of weight $-1$ in blue (see (f)).

(b) Flow $C$ covering all edges of $G$, of size $|C| = \text{width}(G) = 4$ (Lemma 4).

(c) Flow $X + C$.

(d) Unitary flow $D$ matching the parity of $X+C$, i.e., $D \equiv_2 X+C$ (Lemma 5).

(e) Flow $A = (C - D + X)/2$ and a decomposition of it into two paths of weight 1.

(f) Flow $B = (C - D - X)/2$ and a decomposition of it into two paths of weight 1.

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 $\pm 1$ (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 $X : E \to \mathbb{Z}$, $\|X\| \leq 1$, there exist flows $A, B : E \to \mathbb{Z}$ 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 $C - D \equiv X \equiv_2 0$. So we can take $A = (C - D + X)/2$ and $B = (C - D - X)/2$.

1. Notice that $C - D \geq C - 2$ since $\|D\|, \|X\| \leq 1$. So, $C - D \equiv X \equiv_2 0$, since $C \geq 1$.

But $C - D \equiv_2 X \equiv_2 0$ so $C - D \equiv X \equiv_2 0$, whence $A, B \geq 0$.

2. $|A| + |B| = |A + B| = \frac{|C - D + X|}{2} + \frac{|C - D - X|}{2} = |C - D| = |C| - |D| \leq |C|$ since $|D| \geq 0$.

3. And $|C| = \text{width}(G)$.

Corollary 8. For any flow $X : E \to \mathbb{Z}$ with $\|X\| \leq 1$, there exist paths $P_1, \ldots, P_k$ with $k \leq \text{width}(G)$ such that $X = P_1 + \cdots + P_k - P_{k+1} - \cdots - P_k$ (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 $P_1, \ldots, P_{|A|+|B|}$ such that $A = P_1 + \cdots + P_{|A|}$ and $B = P_{|A|+1} + \cdots + P_{|A|+|B|}$. Since $X = A - B$, we can write $X = P_1 + \cdots + P_{|A|} - P_{|A|+1} - \cdots - P_{|A|+|B|}$. 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 $X : E \to \mathbb{Z}$, there exist paths $P_1, \ldots, P_k$ and weights $\{w_1, \ldots, w_k\} \subseteq \{\pm 2^i \mid i \in \mathbb{N}\}$, with $k \leq (\lceil \log \|X\| \rceil + 1) \cdot \text{width}(G)$ such that $X = w_1P_1 + \cdots + w_kP_k$.

Proof. Combine Corollaries 6 and 8, getting

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

where $k_i \leq \text{width}(G)$.

The proof of Theorem 9 suggests a straightforward algorithm for MFD$_\mathbb{Z}$, which we detail in Algorithm 2 and describe at a high level here. First, iteratively decompose $X$, yielding $\lceil \log \|X\| \rceil + 1$ unitary flows. Then use Lemma 7 to decompose each into width($G$) paths. However, as we explained at the beginning of this section, width($G$) is not a lower bound on 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 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 mfd$_\mathbb{Z}(G, X)$. Moreover, the size of this path cover is exactly 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 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 $\log \|X\| + 1$-approximation for 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 $\log \|X\| + 1$ 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 $\log \|X\| + 1$ $Y_i$’s, the for loop on line 14 executes at most $\log \|X\| + 1$ 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 $O(m \log \|X\| \cdot \text{width}(G') + n) = O(m \log \|X\| \cdot \text{mfd}_2(G, X) + n)$.

A theorem analogous to Theorem 9 for MFD$_N$ 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 MFD$_N$ 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 MFD$_N$. Indeed, the particular greedy path removal strategy of finding the heaviest path (heaviest-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 MFD$_N$

We define a family of MFD$_N$ 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 $\ell$. 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 $\ell$, 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: $\mathcal{P} \leftarrow \emptyset$, $w \leftarrow [0]$  ▶ length-zero vectors
5: $C \leftarrow \text{flow of value width}(G')$, $C \geq 1$  ▶ 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 $j \in \{0, \ldots, i\}$ 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 $G_\ell$ with flow $X_{G_\ell,2^{\ell+1}}$ be constructed as described before. Greedy-weight uses $1 + 2^{\ell + 1}$ paths to decompose $X_{G_\ell,2^{\ell+1}}$.

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

- Theorem 14. The approximation ratio for greedy-weight on MFD$_\exists$ 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 $X_{G_\ell,2^{\ell+1}}$ 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 $G_\ell$ 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 $(G_\ell, X_{G_\ell,2^{\ell+1}})$ so that this approach has the same approximation ratio, consider $(G_\ell^*, X_{G_\ell^*,2^{\ell+1}})$, constructed as in $(G_\ell, X_{G_\ell,2^{\ell+1}})$ 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
As exploited in Section 4.1, one sticking point for greedy path removal algorithms is the fact that MFD's NP-hardness 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). \( G_t^\ast, X_{t,2^{t+1}}^\ast \) will also force this approach to take an exponential number of paths, since first the algorithm will take all \( 2^{t+1} \) weight-1 edges with \( 2^{t+1} \) 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 MFD_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 MFD_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 MFD_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 MFD_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].

\[ \text{Lemma 17. Let } G = (V,E) \text{ be a graph, width}(G) \geq 2, \text{ satisfying Properties 15 and 16. Greedy-weight uses at most } \left\lceil \log |X| / \log \frac{\text{width}(G)}{b} \right\rceil + 1 \text{ paths to decompose any flow } X : E \to \mathbb{N}. \]

**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 \( c = \left\lceil \log |X| / \log \frac{b}{b-1} \right\rceil + 1 \) steps (paths).

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

**Proof.** Assume \( X > 0 \) (otherwise, replace \( G \) with \( G_{\{X\}} \)). Thus, \( b = \text{width}(G) \leq \text{mfd}_N(G,X) \), 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 an \( O\left( \frac{\log |X|}{b \log \frac{b}{b-1}} \right) = O(\log |X|) \)-approximation for \( \text{MFD}_N \) \((b \log \frac{b}{b-1} = O(1) \) 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 an \( O(\log |X|) \)-approximation for \( \text{MFD}_N \) 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 \( X : E \to \mathbb{Y} \) on a graph \( G = (V,E) \) and a set of \( s \)-\( t \) paths \( \{P_1, \ldots, P_k\} \), 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 \( k \text{-FWA}_Y \) 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 \( X : E \to \mathbb{Y} \) 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 \( L_X = L_Y(P_1, \ldots, P_k) = \{w \in \mathbb{Y}^k | X = \sum_{j=1}^k P_j w_j \} \) the linear system corresponding to the paths \( P_1, \ldots, P_k \).
Width Helps and Hinders Splitting Flows

We shortly discuss how to solve $k$-FWA$_N$. 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 $k$-FWA$_N$ turns out to be more difficult. Here, the linear system contains the inequality $w \geq 0$. In fact, it was shown [14] that $k$-FWA$_N$ is NP-hard. The program Toboggan [14] implements a linear FPT algorithm for MFD$_N$. One step of the algorithm is to solve $k$-FWA$_N$ using an ILP [14]. The authors state the following conjecture.

**Conjecture 22 ([14]).** If $(P_1, \ldots, P_k)$ are the paths of a minimum flow decomposition of $(G, X)$, then the linear system $L_N(P_1, \ldots, P_k)$ 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 $k$-FWA$_N$ from Figure 4 and the solution therein. We now give another solution for $k$-FWA$_N$ 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$_N$ 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 MFD$_N$ [7] on this instance, both returning $\text{mfd}_N(G, X) = 8$.

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 (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 $A_iB_j$ 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 $\lambda_1$ and $\lambda_2$ on the edges labeled $A_0B_0$. 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 $\lambda_1$ by some $\delta > 0$, the weights of $B_1$ and $B_2$ each increase by $\delta/2$. And thus, $\delta$ must be even. Due to the parity of $a_0$ and $b_0$, they can never reach 0.

**Figure 4** Paths $A_i$ and $B_i$ (i ∈ {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 $X = \sum_{i=0}^{3} a_i A_i + \sum_{i=0}^{2} b_i B_i$. Note that these weights are a solution of $k$-FWA$_N$ on input $(G, X)$ with given paths $A_i, B_i$ (i ∈ {0, 1, 2, 3}).
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 MFD$_Z$ 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 MFD$_N$.

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


