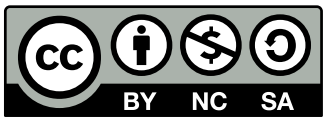
Aalto University
School of Science

Master's Programme in Mathematics and Operations Research

Decomposing flow networks into a minimum-size set of paths with subpath constraints

Antti Immonen

Master's Thesis
2026



Aalto University
School of Science

Author Antti Immonen

Title Decomposing flow networks into a minimum-size set of paths with subpath constraints

Degree programme Mathematics and Operations Research

Major Mathematics

Supervisor Assistant Prof. Philine Schiewe

Advisor Dr. Fernando Dias

Date May 25, 2026 **Number of pages** 69 **Language** English

Abstract

The minimum flow decomposition (MFD) problem asks for the smallest collection of weighted paths whose weights together reproduce a given network flow. The problem arises in computational biology, where the flow represents read coverage in RNA transcript assembly. Two extensions of MFD have recently been studied: restricting the path weights to a fixed set W (MFDW), which constrains the search space, and forcing a given collection of subpaths to appear in the solution (MFDSC), which captures information from long-read and paired-end sequencing. Real RNA-seq instances typically exhibit both features: coverage values cluster around a few abundance levels, and long reads supply subpath constraints.

This thesis studies the combined problem, MFDW-SC, and develops an exact solver for it. The first contribution is a compact linear path-indexed integer program (ILP) that avoids the nonlinear weight-edge products of the standard MFD formulation by indexing paths directly by their weights from the finite set W. The second and main contribution is a structural decomposition based on weight equivalence classes and layer graphs that partitions W into groups of weights inducing the same topology. From this decomposition, we derive a weighted antichain lower bound on the number of paths in any feasible solution, built from a per-edge integer-decomposition number $\pi(e)$. The bound dominates the unweighted antichain bound of Cáceres et al. [1] and gives a single starting value k_0 for the iterative search. The same structural view lets us remove provably infeasible variables before the ILP is built and add a few supporting inequalities suggested by the layered model.

We evaluate the resulting Layered ILP solver on splice graphs from RNA sequencing data. It returns the same optima as an equivalent baseline integer program while solving consistently faster. The further inequalities suggested by the layered model are reported as a separate ablation, since they do not improve runtime on this benchmark.

Keywords Optimization, network flow, flow decomposition, integer programming, subpath constraints, RNA assembly

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Antti Immonen

Työn nimi Virtausverkkojen hajottaminen minimaaliseksi polkujoukoksi alipolkurajoitteiden avulla

Koulutusohjelma Mathematics and Operations Research

Pääaine Mathematics

Työn valvoja ja ohjaaja Assistant Prof. Philine Schiewe

Päivämäärä 25. toukokuuta 2026 **Sivumäärä** 69 **Kieli** englanti

Tiivistelmä

Minimivirtaushajotelman (MFD) ongelmassa etsitään pienintä painotettujen polkujen joukkoa, jonka painot yhdessä tuottavat annetun verkkovirtauksen. Ongelma nousee esiin laskennallisessa biologiassa, jossa virtaus kuvaa lukukattavuutta RNA-transkriptien kokoamisessa. MFD:lle on viime aikoina tutkittu kahta laajennusta: polkujen painojen rajaamista kiinteään joukkoon W (MFDW), mikä pienentää hakuavaruutta, sekä tiettyjen alipolkujen pakottamista esiintymään ratkaisussa (MFDSC), mikä hyödyntää pitkien lukujen ja paripääsekvensoinnin (paired-end sequencing) tarjoamaa tietoa. Todellisissa RNA-sekvensointiaineistoissa nämä piirteet esiintyvät tyypillisesti yhdessä: lukukattavuuden arvot keskittyvät muutamaan runsaustasoon, ja pitkät lukemat tuottavat alipolkurajoitteita.

Tässä työssä tarkastellaan näiden yhdistelmää, MFDW-SC-ongelmaa, ja kehitetään sille eksakti ratkaisin. Ensimmäisenä esitetään tiivis, lineaarinen polkuindeksoitu kokonaislukuoptimointimalli (integer linear program), joka välttää tavanomaisen MFD-mallin epälineaariset painon ja kaaren tulot indeksoimalla polut suoraan painoillaan äärellisestä joukosta W. Toisena ja keskeisimpänä kehitetään rakenteellinen hajotelma (structural decomposition), joka perustuu painojen ekvivalenssiluokkiin ja kerrosgraafeihin (layer graphs) ja jakaa joukon W ryhmiin, joiden painot indusoivat saman topologian. Tästä hajotelmasta johdetaan painotettu antiketju-alaraja (weighted antichain lower bound) minkä tahansa käyvän ratkaisun polkujen lukumäärälle; alaraja rakentuu kaarikohtaisesta kokonaislukuhajotelmaluvusta (integer-decomposition number) $\pi(e)$. Alaraja on vähintään yhtä tiukka kuin Cáceresin ym. [1] painottamaton antiketju-alaraja ja antaa iteratiiviselle haulle yhden aloitusarvon k_0. Sama rakenteellinen näkökulma mahdollistaa todistetusti epäkäypien muuttujien poistamisen ennen ILP-mallin muodostamista sekä muutaman kerrosmallista (layered model) seuraavan täydentävän epäyhtälön lisäämisen.

Syntyvää ILP-ratkaisinta arvioidaan RNA-sekvensoinnista peräisin olevilla silmukointigraafeilla (splice graphs). Se tuottaa samat optimit kuin vastaava perustason kokonaislukumalli mutta ratkaisee ne johdonmukaisesti nopeammin. Kerrosmallista (layered model) johdettavat lisäepäyhtälöt raportoidaan erillisenä lisätarkasteluna, sillä ne eivät nopeuta ratkaisua tällä vertailuaineistolla.

Avainsanat optimointi, verkkovirtaus, virtaushajotelma, kokonaislukuoptimointi, alipolkurajoitteet, transkriptien kokoaminen

Preface

This thesis was completed as part of my Master's degree in Mathematics at Aalto University. The work builds upon recent advances in flow decomposition algorithms for computational biology, particularly the ILP formulations of Dias et al. [3] and the weight-restricted approach of Grigorjew et al. [4].

Espoo, May 25, 2026

Antti Immonen

Contents

Symbols and abbreviations

Font styles

$a, b, c, \ldots$	an element
$A, B, C, \ldots$	a set
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$	a collection of sets
`ALGORITHM`	algorithm name in monospaced font

Symbols

$G = (V, E)$	directed acyclic graph with source s and sink t
$f : E \to \mathbb{Z}_{>0}$	integral flow on edges
$T = \{t_1 < \cdots < t_q\}$	distinct values in $\{f(e) : e \in E\}$
$E_w = \{e \in E : f(e) \geq w\}$, $G_w = (V, E_w)$	weight-w layer graph
$[w] = \min\{t \in T : t \geq w\}$	representative value of weight w
$C_r = \{w \in W : [w] = r\}$	weight class with representative $r \in T \cup \{+\infty\}$
R_j	required subpath (sequence of edges)
$s(R_j), t(R_j)$	first and last vertex of R_j
$\Pi_{R_j}(G)$	"there exists an s-t path in G that contains R_j"
$W_{\text{topo}}(R_j)$	weights whose layer admits a path $s \to R_j \to t$
$U_{R_j} = \min_{e \in R_j} f(e)$	bottleneck weight of R_j
$W_{\text{valid}}(R_j)$	$W_{\text{topo}}(R_j) \cap \{w \in W : w \leq U_{R_j}\}$
$W_{\text{valid}}(u, v)$	weights $w \in W$ with $w \leq f(u, v)$ and $u \in \mathcal{S}_{[w]}$, $v \in \mathcal{T}_{[w]}$
$i \in [k]$	path slot index
$\ell \in [\lvert W \rvert]$	weight index ($w_\ell \in W$)
$j \in [m]$	constraint index
$z_{uvi\ell} \in \{0, 1\}$	edge (u, v) used by slot i with weight w_ℓ
$y_{i\ell} \in \{0, 1\}$	slot i is active with weight w_ℓ
$r_{ij} \in \{0, 1\}$	slot i contains subpath R_j
k_0	starting lower bound on #paths
$\mathcal{S}_r$	set of vertices reachable from source s in layer G_r
$\mathcal{T}_r$	set of vertices that can reach sink t in layer G_r

Abbreviations

MFD	Minimum Flow Decomposition
MFDW	MFD with Given Weights
MFDSC	MFD with Subpath Constraints
MFDW-SC	MFD with Given Weights and Subpath Constraints
ILP	Integer Linear Programming
LP	Linear Programming
FPT	Fixed-Parameter Tractable
RNA-seq	RNA Sequencing
DAG	Directed Acyclic Graph
BFS	Breadth-First Search
DFS	Depth-First Search
ECC	Edge Conflict Cut
EDC	Edge Integer-Decomposition Cut

1 Introduction

RNA-sequencing has become an essential tool for studying gene expression and regulation [15, 13]. Unlike DNA, which provides the static blueprint of an organism, RNA reflects which genes are active under given cellular conditions. Through a process called *alternative splicing* a single gene can produce multiple distinct RNA transcripts called *isoforms*, each potentially encoding a different protein. Modern sequencing technologies produce millions of short reads sampled from the RNA molecules in a biological sample, and the computational challenge of reconstructing the full-length transcripts and their abundances from these fragments is known as *transcript assembly*.

A common method used by tools such as StringTie [9] and Scallop [11] represents the assembly problem with a *splice graph*, a directed acyclic graph in which each vertex is an exon, each edge is an exon adjacency or junction and each edge weight $f(e)$ counts the reads supporting that connection. A path from source to sink is a candidate transcript and the path weight estimates its abundance. Under uniform read sampling and no sequencing errors the edge weights form a *flow*, so transcript assembly reduces to decomposing that flow into weighted source-to-sink paths. Any flow admits many such decompositions, but for well-assembled genes the minimum decomposition typically matches the true transcript set [6], which motivates the *Minimum Flow Decomposition* (MFD) problem of finding the smallest number of weighted paths consistent with the edge flow. The formal definition is deferred to Section 2.

MFD is NP-hard on DAGs [5] and recent work focuses on practical solvers that exploit small solution sizes. Two extensions are particularly relevant for RNA assembly. Reads produced by long-read technologies such as PacBio [10] and Oxford Nanopore [2], together with paired-end Illumina reads, often span multiple exons and reveal which junctions co-occur on the same transcript; encoding these as a collection of required subpaths gives the *MFDSC* problem of Williams et al. [16], who showed that the added constraints improve assembly accuracy. Independently, transcript abundances cluster around a handful of values rather than spanning the integers freely, with the read-count distribution per gene typically spanning a few orders of magnitude between highly and weakly expressed isoforms [13]. Restricting path weights to a fixed finite set W gives the *MFDW* problem of Grigorjew et al. [4], whose main observation is that a well-chosen W accelerates the ILP without inflating the number of paths. Real datasets exhibit both properties, yet no method exploits them jointly.

Can we combine weight restrictions and subpath constraints to obtain speedups beyond either alone, while still solving the resulting problem exactly?

We introduce the *MFDW-SC problem*: a minimum flow decomposition where path weights come from a given set W and all subpath constraints in $\mathcal{R}$ are satisfied.

Contributions

The thesis makes three principal contributions. First, we give a formal definition of MFDW-SC together with a fully linear path-indexed ILP that avoids the nonlinear weight-edge products of the standard MFD formulation by indexing path slots directly by their weight from the finite set W (Section 3). Second, we develop a structural framework based on weight equivalence classes C_r, layer graphs G_r and the feasible weight set W_{valid}, which together form the layered model that all subsequent refinements build on (Section 4). Third, we derive a weighted antichain lower bound from a per-edge integer-decomposition number $\pi(e)$: for any edge antichain $\mathcal{A}$, every feasible MFDW-SC decomposition uses at least $\sum_{e \in \mathcal{A}} \pi(e)$ paths, which is at least as tight as both the unweighted antichain bound of Cáceres et al. [1] and the per-edge ceiling $\lceil f(e)/\max W_{\text{valid}}(e) \rceil$ and strictly tighter whenever some $\pi(e) > 1$ (Theorem 4.5).

The layered model removes provably infeasible $z_{uvi\ell}$ variables before the ILP is built, with mean variable reduction of 72.4% on the benchmark (Section 5). The full solver is a four-phase pipeline that combines the layered model with Y-to-V graph reduction, safe-path identification and a branching priority on the most restricted cover variables, then runs the iterative search starting from the bound of Theorem 4.5.

The layered model also suggests several additional valid inequalities (edge conflict cuts, constraint incompatibility cuts, multiplicity upper bounds, edge integer-decomposition cuts and forced single-path constraints). We prove their validity and report them in a separate ablation in Section 6; their practical effect is mixed, so they are not enabled in the recommended solver configuration.

1.1 Thesis organization

Section 2 covers the background on graph theory, flow networks, computational complexity and related work on flow decomposition. Section 3 defines the MFDW-SC problem, presents the path-indexed ILP and establishes its computational complexity. Section 4 develops the structural theory: weight equivalence classes, layer graphs, the per-edge integer-decomposition number $\pi(e)$, the weighted antichain lower bound and the additional valid inequalities that the layered model suggests. Section 5 describes the solver implementation. Section 6 presents the experimental evaluation and identifies which components pay off in practice on real RNA splice graphs.

2 Background

We fix notation for directed graphs and paths before introducing flow networks. A *directed graph* $G = (V, E)$ has a finite vertex set V and an edge set $E \subseteq V \times V$; we write $(u, v) \in E$ for an edge from u to v.

A *directed cycle* is a sequence of vertices $v_0, v_1, \ldots, v_\ell = v_0$ with $\ell \geq 1$ and $(v_{i-1}, v_i) \in E$ for every i. The graph is a *directed acyclic graph* (DAG) if it contains no directed cycle. For a vertex v, the *in-degree* is $\text{in-deg}(v) = |\{(u, v) \in E\}|$ and the *out-degree* is $\text{out-deg}(v) = |\{(v, u') \in E\}|$. A vertex s with in-degree zero is a *source*, and a vertex t with out-degree zero is a *sink*.

A *path* P in G is a sequence of vertices $P = (v_0, v_1, \ldots, v_\ell)$ with $(v_i, v_{i+1}) \in E$ for all i, equivalently the edge sequence $(e_1, \ldots, e_\ell)$ with $e_i = (v_{i-1}, v_i)$. Its *length* is ℓ. It is *simple* if all vertices are distinct, and an *s-t path* runs from s to t. We write $a \rightsquigarrow b$ when G contains a path from a to b and identify P with its edge set when convenient.

A path $Q = (e'_1, \ldots, e'_p)$ is a *subpath* of $P = (e_1, \ldots, e_\ell)$, written $Q \subseteq P$, if its edges appear as a contiguous subsequence of P: there exists $1 \leq i \leq \ell - p + 1$ such that $e'_j = e_{i+j-1}$ for all j.

Definition 2.1 (Path independence)**.** Two paths P_1, P_2 in a DAG are *independent* if no path P^* contains both as subpaths.

2.1 Flow networks

Definition 2.2 (Flow Network)**.** A *flow network* is a tuple $G = (V, E, f)$, where:

- (V, E) is a directed acyclic graph (DAG) with unique source vertex s and unique sink vertex t;
- $f : E \to \mathbb{Z}_{>0}$ is a *flow function* assigning positive integer flow values to edges;
- For all vertices $v \in V \setminus \{s, t\}$, the flow satisfies *conservation*:

$$\sum_{u:(u,v)\in E} f(u, v) = \sum_{u':(v,u')\in E} f(v, u'). \tag{1}$$

2.2 Flow decomposition

Definition 2.3 (Flow Decomposition)**.** A *k-flow decomposition* of a flow network $G = (V, E, f)$ is a pair $(\mathcal{P}, w)$ consisting of:

- A sequence of k source-to-sink paths $\mathcal{P} = (P_1, \ldots, P_k)$, where each P_i is an s-t path;
- A weight vector $w = (w_1, \ldots, w_k) \in \mathbb{Z}^k_{>0}$ of positive integer weights;

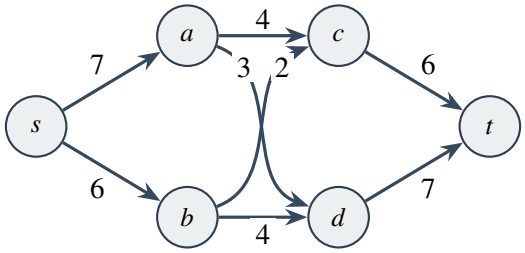


Figure 1: A flow network $G = (V, E, f)$ with source s, sink t and integer flow values on edges. This network is the running example reused for the path decomposition in Figure 2 and the edge-antichain illustrations in Figure 3.

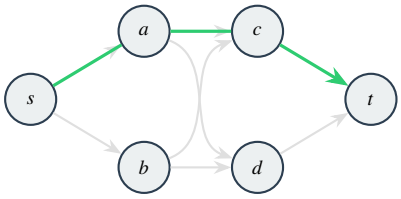
such that for every edge $e = (u, v) \in E$:

$$\sum_{i \in \{1,\dots,k\}\,;\, e \in P_i} w_i = f(e). \tag{2}$$

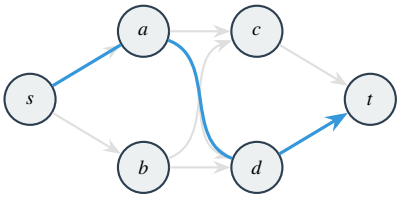
We call $k = |\mathcal{P}|$ the *size* of the decomposition. Figure 1 shows a small flow network and Figure 2 displays one of its decompositions.

Proposition 2.1 ([14])**.** *Every flow network $G = (V, E, f)$ admits a flow decomposition with at most $|E| - |V| + 2$ paths.*

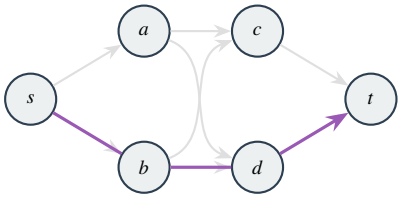
Lower bounds on the size of any flow decomposition will be central to the solver developed later. The simplest of these comes from *edge antichains*: a set $\mathcal{A} \subseteq E$ of edges is an *antichain* if no s-t path contains two distinct edges of $\mathcal{A}$. Figure 3 illustrates pairs of edges that are or are not independent in this sense.



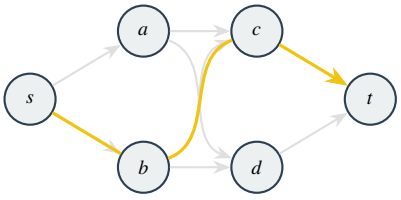
(a) $P_1 = (s, a, c, t)$, $w_1 = 4$



(b) $P_2 = (s, a, d, t)$, $w_2 = 3$



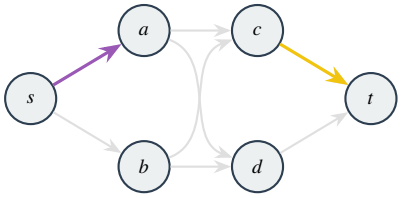
(c) $P_3 = (s, b, d, t)$, $w_3 = 4$



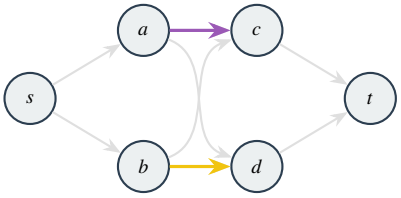
(d) $P_4 = (s, b, c, t)$, $w_4 = 2$

Figure 2: A four-path decomposition of the flow network in Figure 1. Each subfigure highlights one path of the decomposition with its assigned weight; the remaining edges are faded for clarity. Summing the highlighted weights at each edge reproduces the flow values shown in Figure 1.

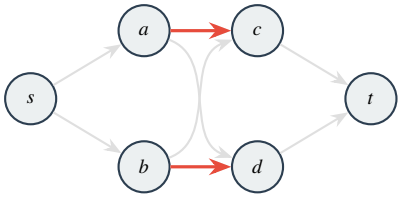
Proposition 2.2 ([1])**.** *For any flow network $G = (V, E, f)$ and any edge antichain $\mathcal{A} \subseteq E$, $|\mathcal{A}|$ is a lower bound on the size of any flow decomposition (see Figure 4).*



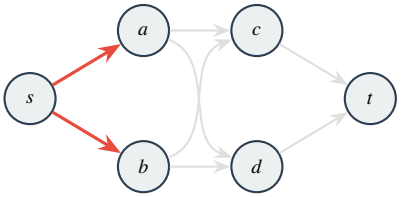
(a) Dependent edges: (s, a) (purple) and (c, t) (yellow). A single path can contain both.



(b) Independent edges: (a, c) (purple) and (b, d) (yellow). No single s-t path can contain both.



(c) An antichain: $\{(a, c), (b, d)\}$ (red). Both edges are independent. Size = 2, so $k \geq 2$.



(d) Another antichain: $\{(s, a), (s, b)\}$ (red). Edges leaving the source are trivially independent. Size = 2, so $k \geq 2$.

Figure 3: Edge independence and antichains in a small DAG. Pairs (a) and (b) contrast a dependent pair (some s-t path contains both) with an independent pair (no s-t path contains both). Subfigures (c) and (d) show two antichains of size two, each giving a lower bound of $k \geq 2$ on any flow decomposition of the network in Figure 1 via Proposition 2.2.

Proof. Since edges in $\mathcal{A}$ are pairwise independent, no single path in a decomposition can traverse more than one edge from $\mathcal{A}$. Therefore, at least $|\mathcal{A}|$ distinct paths are required to cover all edges in $\mathcal{A}$. □

A *decision problem* has a binary answer. The class P contains decision problems solvable in polynomial time, and NP contains problems for which a candidate solution can be verified in polynomial time. A problem is *NP-hard* if every problem in NP reduces to it in polynomial time, and *NP-complete* if it is both NP-hard and in NP. Polynomial-time reduction is the standard tool for proving NP-hardness: problem A reduces to problem B, written $A \leq_P B$, if there is a polynomial-time computable map f such that x is a yes-instance of A if and only if $f(x)$ is a yes-instance of B.

Hartman et al. [5] showed NP-hardness of MFD by reduction from the 3-*Partition* problem: given $3m$ positive integers that sum to mB with each between $B/4$ and $B/2$, decide whether they partition into m triples that each sum to B. Given such a 3-Partition instance, the reduction constructs a small DAG in which a flow decomposition of size m exists if and only if the partition exists; the integers become path weights and the m groups become m paths.

Two relaxations of the worst-case picture are relevant later in the thesis. An algorithm is *fixed-parameter tractable* (FPT) with respect to a parameter k if its running time is $f(k) \cdot \text{poly}(n)$, where f depends only on k and n denotes the input size. FPT algorithms are useful whenever k is small in practice, even if f grows like 2^{k^2}. On the approximation side, an algorithm for a minimization problem has

approximation ratio α if it always returns a solution of value at most $\alpha \cdot \mathrm{OPT}$. For MFD, the parity-balancing algorithm of Mumey et al. [8] gives an approximation ratio polylogarithmic in $\|f\|_\infty = \max_{e \in E} f(e)$, and the powers-of-two weight set used by Grigorjew et al. [4] yields a $(\lceil \log_2 \|f\|_\infty \rceil + 1)$-approximation by reduction to MFDW.

An *integer linear program* (ILP) minimizes or maximizes a linear objective $c^\top x$ subject to linear constraints $Ax \le b$ with $x \in \mathbb{Z}^n$, given $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$.

Despite being NP-hard in general, modern ILP solvers like Gurobi perform well on practical instances through branch-and-bound, cutting planes and presolving. The bilinear product between an edge indicator $x_{uvi} \in \{0, 1\}$ and a bounded path weight w_i is the standard difficulty in flow-decomposition formulations; we record the big-M linearization of this product in the example below.

The basic ILP formulation for k-flow decomposition is given below; it is the basis for the extensions developed later.

Example 2.1. Given a flow network $G = (V, E, f)$ and a parameter k, determine whether there exists a decomposition of the flow into exactly k weighted paths.

The formulation uses binary edge indicators $x_{uvi} \in \{0, 1\}$ for each edge $(u, v) \in E$ and path index $i \in \{1, \dots, k\}$ (with $x_{uvi} = 1$ when edge (u, v) belongs to path i), integer path weights $w_i \in \mathbb{Z}_{>0}$ and auxiliary variables $\mu_{uvi} \in \mathbb{Z}_{\ge 0}$ used to linearize the product $x_{uvi} \cdot w_i$.

The path structure constraints force each path i to be a valid s-t path:

$$\sum_{(s,v) \in E} x_{svi} = 1, \quad \forall i \in \{1, \dots, k\} \qquad \text{(source)}$$

$$\sum_{(u,t) \in E} x_{uti} = 1, \quad \forall i \in \{1, \dots, k\} \qquad \text{(sink)}$$

$$\sum_{(u,v) \in E} x_{uvi} - \sum_{(v,u') \in E} x_{vu'i} = 0, \quad \forall v \in V \setminus \{s, t\}, \forall i \qquad \text{(conservation)}$$

The flow decomposition constraint enforces that the weighted paths reproduce the input flow,

$$\sum_{i=1}^{k} \mu_{uvi} = f(u, v), \quad \forall (u, v) \in E, \tag{3}$$

and the product $x_{uvi} \cdot w_i$ is linearized through the auxiliary variables μ_{uvi} with $M = \max_{e \in E} f(e)$:

$$\begin{aligned} &\mu_{uvi} \le M \cdot x_{uvi}, \\ &\mu_{uvi} \le w_i, \\ &\mu_{uvi} \ge w_i - M(1 - x_{uvi}). \end{aligned}$$

Together these enforce $\mu_{uvi} = w_i$ when $x_{uvi} = 1$ and $\mu_{uvi} = 0$ otherwise. To find the minimum flow decomposition, we test feasibility for $k = 1, 2, 3, \dots$ until the ILP first becomes feasible.

2.3 Related work on flow decomposition

The MFD problem was formalized by Vatinlen et al. [14], who proved the $|E| - |V| + 2$ upper bound on decomposition size and proposed greedy heuristics. Hartman et al. [5] subsequently proved that MFD is NP-hard even on DAGs, establishing the computational difficulty of the problem.

The first practical exact algorithms for MFD came from two directions. Kloster et al. [7] developed Toboggan, an FPT algorithm running in time $2^{O(k^2)} \cdot \text{poly}(n)$ where k is the solution size. Their approach combines graph reduction via Y-to-V contraction of degree-1 nodes with bounded search trees and branching rules, solving instances with $k \leq 6$ quickly but becoming impractical for larger k. Dias et al. [3] used an ILP, encoding paths via flow conservation constraints rather than enumerating all paths. Their formulation uses $O(k \cdot |E|)$ variables and constraints and finishes within a few seconds on every instance of their benchmark within a one-minute timeout. On the heuristic side, Shao and Kingsford [12] proposed Catfish, a greedy algorithm that iteratively selects the path with maximum bottleneck flow. Catfish has no proven approximation ratio, but solves any instance in seconds and is competitive on easy instances.

Williams et al. [16] introduced MFDSC to incorporate long-read constraints into flow decomposition. They extended the Toboggan FPT algorithm into Coaster, but found it practical only for small k on benchmark instances. Their ILP extension adds binary variables $r_{ij} \in \{0, 1\}$ indicating whether path i satisfies constraint R_j, together with covering constraints ensuring that each constraint is satisfied by at least one path.

Grigorjew et al. [4] introduced MFDW, showing that restricting path weights to a finite set W allows the ILP to be simplified. Instead of treating the path weight w_i as an integer variable, they use weight-indexed variables $x_{uvi}^{(k)} \in \mathbb{Z}_{>0}$ counting the flow of weight w_k on edge (u, v) in path i. With powers of two as the weight set, $W = \{2^k \mid 0 \leq k \leq \lceil \log \|f\|_\infty \rceil\}$, their solution achieves a $(\lceil \log \|f\|_\infty \rceil + 1)$-approximation. Augmenting this with observed flow values, $W = \{2^k \mid k \in \mathbb{Z}_{\geq 0}\} \cup \{f(e) \mid e \in E\}$, matches the unrestricted MFD solution on most instances, with average speedups of roughly 50 to 70× over the exact MFD solver and up to about 90× on the hardest instances, rising further to roughly 120× on the subpath-constrained variant.

2.4 Practical ILP optimization techniques

Several known techniques accelerate ILP-based flow decomposition solvers; we review the ones most relevant to our approach.

As noted in Proposition 2.2, edge antichains provide lower bounds on the optimal decomposition size $k^\star$, defined as the smallest k for which a feasible decomposition exists. This serves two purposes: starting the ILP iteration at $k = |A|$ instead of $k = 1$, and certifying heuristic optimality without running the ILP.

Proposition 2.3. *If a heuristic finds a decomposition of size k and we can compute an antichain of size k, then the heuristic solution is optimal.*

Proof. Immediate from Proposition 2.2: the antichain forces $k \leq k^\star$ while the heuristic gives $k^\star \leq k$. □

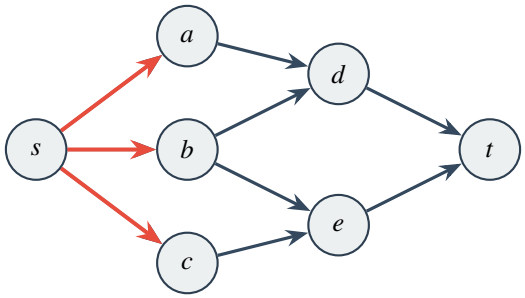


Figure 4: An antichain of three edges (highlighted in red). Since these edges are pairwise independent (no path contains two of them), any flow decomposition requires at least 3 paths. This provides a lower bound: $k^\star \geq 3$.

ILP solvers can waste time exploring symmetric solutions (e.g., path orderings). A standard technique is to impose an ordering on path weights:

$$w_1 \geq w_2 \geq \cdots \geq w_k$$

This constraint eliminates factorial redundancy without affecting the optimal value of k.

When multiple subpath constraints are present, two or more of them may be satisfiable by a single solution path or only by separate paths. Williams et al. [16] formalized this with the notion of *constraint compatibility*.

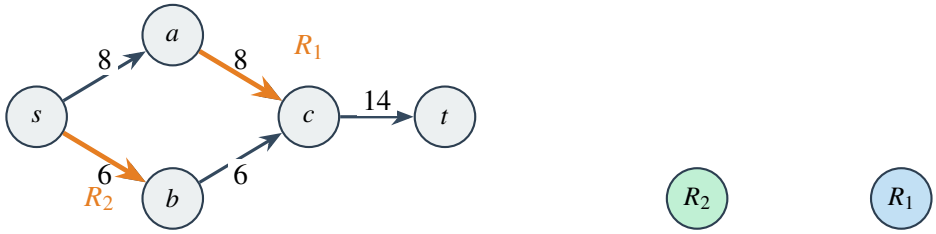
Definition 2.4 (Constraint compatibility)**.** Two subpath constraints R_i and R_j are *compatible* if there exists an s-t path P in G such that both $R_i \subseteq P$ and $R_j \subseteq P$.

Compatibility is determined by the topological relationship between constraints. If R_i ends before R_j begins (i.e., the last vertex of R_i can reach the first vertex of R_j), they may be compatible. Conversely, if R_i and R_j lie on topologically disjoint branches, or if they overlap or conflict in their edge sequences, then they are incompatible.

Definition 2.5 (Constraint graph)**.** The *constraint graph* $G^c = (\mathcal{R}, E^c)$ is a directed graph whose vertices are the subpath constraints $V(G^c) = \{R_1, \ldots, R_m\}$ and whose edges connect compatible pairs: $(R_i, R_j) \in E^c$ if R_i and R_j are compatible and R_i can precede R_j in an s-t path.

Figure 5 shows a small example where two constraints lie on disjoint branches and therefore cannot share a solution path. The constraint graph provides a lower bound on the solution size: if G^c has k vertex-disjoint paths, then at least k solution paths are needed [16]. This bound can be computed efficiently via maximum antichain algorithms on the constraint graph.

In our setting with restricted weights, two constraints R_i and R_j can share a single path of weight w only if they are topologically compatible in G, the weight $w \in W$ is admissible for both constraints and the subgraph formed by edges with



(a) Flow network with constraints $R_1 = (a, c)$ and $R_2 = (s, b)$

(b) Constraint graph with no edges (constraints are incompatible)

Figure 5: Constraint compatibility example. In (a), constraints $R_2 = (s, b)$ and $R_1 = (a, c)$ are topologically incompatible because they lie on disjoint branches (upper through a, lower through b). The constraint graph in (b) therefore has no edges, requiring at least 2 paths. If both constraints were on the same branch, a directed edge $R_2 \rightarrow R_1$ would indicate they could be merged into a single solution path.

flow at least w still contains an s-t path that traverses both. This restriction can break compatibility relationships that would exist in unrestricted MFDSC, which is one reason weight-aware preprocessing helps in the combined problem.

The approach used throughout this thesis is to separate optimization over the number of paths from feasibility for a fixed value of k. For a chosen k, we ask whether there exists a k-path decomposition whose weights lie in W and whose paths cover all constraints in $\mathcal{R}$. The smallest feasible value of k is then the optimum.

Separating optimization over k from fixed-k feasibility matches both features that motivate MFDW-SC. A finite weight set turns the integer-weight search into a discrete enumeration, which leads to a compact fixed-k formulation. Subpath constraints become a covering question at fixed k: can all required subpaths be routed through at most k solution paths? Combining the two gives a feasibility problem in which structural preprocessing can rule out impossible weight-edge-constraint combinations before the ILP is built.

In practice we start the search from a structural lower bound k_0 rather than from $k = 1$, so the loop tests $k = k_0, k_0 + 1, \ldots$ until the first feasible value is reached. If a heuristic already produces a decomposition of size k_h matching the lower bound, optimality is certified without solving any ILP. Section 6 compares this iterative scheme against an auxiliary single-solve mode that builds one larger model with a fixed path cap.

3 Problem formulation and ILP

3.1 The MFDW-SC problem and its ILP

Definition 3.1 below states the problem combining weight restrictions and subpath constraints.

Definition 3.1 (MFDW-SC problem)**.** An instance of the MFDW-SC problem is a tuple $I = (G, f, W, \mathcal{R})$, where $G = (V, E)$ is a directed acyclic graph with unique source s and sink t, $f : E \to \mathbb{Z}_{>0}$ is a flow function satisfying conservation, $W = \{w_1, \ldots, w_{|W|}\} \subset \mathbb{Z}_{>0}$ is a finite set of allowed path weights and $\mathcal{R} = \{R_1, \ldots, R_m\}$ is a set of subpath constraints with each $R_j = (e^j_1, \ldots, e^j_{p_j})$ a directed path in G.

A feasible solution is a set of path-weight pairs $\mathcal{S} = \{(P_1, \tilde{w}_1), \ldots, (P_k, \tilde{w}_k)\}$ where each P_i is an s-t path and each $\tilde{w}_i \in W$, such that

$$\sum_{i:e\in P_i} \tilde{w}_i = f(e) \qquad \forall e \in E,$$

and for every $R_j \in \mathcal{R}$ there exists a path P_i that contains R_j as a subpath. The objective is to minimize $k = |\mathcal{S}|$.

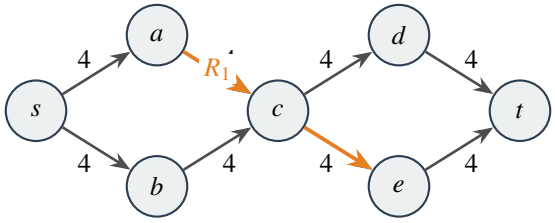
When a graph is fixed, we sometimes write a subpath by its vertex sequence, for example $R_1 = (a, c, e)$. This abbreviates the corresponding edge sequence $((a, c), (c, e))$.

Example 3.1. Figure 6 introduces the running instance: every edge carries flow 4, $W = \{2, 4\}$ and $R_1 = (a, c, e)$. Of the two valid 2-path decompositions, only the pairing that routes the a-side through e contains R_1, so the unique optimal MFDW-SC solution is

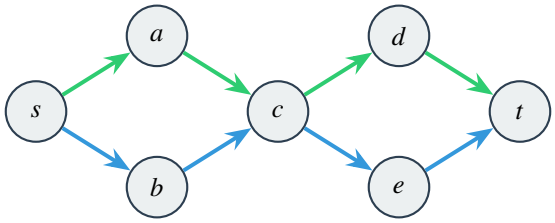
$$\begin{aligned} P_1 &= (s, a, c, e, t) \quad \text{with weight } 4 \quad (\text{satisfies } R_1), \\ P_2 &= (s, b, c, d, t) \quad \text{with weight } 4. \end{aligned}$$

Adding a second constraint $R_2 = (b, c, e)$ requires both an a-route and a b-route through c to e. Because no s-t path uses both (a, c) and (b, c), two distinct paths must traverse (c, e); the only way to fit them in $W = \{2, 4\}$ within the bottleneck flow $f(c, e) = 4$ is to use weight 2 each. The remaining flow on (s, a), (s, b), (c, d), (d, t) then requires two further weight-2 paths through (c, d), raising the optimum from 2 to 4 paths with no change to the edge flows. The constraint set determines which routings remain feasible and the weight set W determines whether those routings admit a small decomposition.

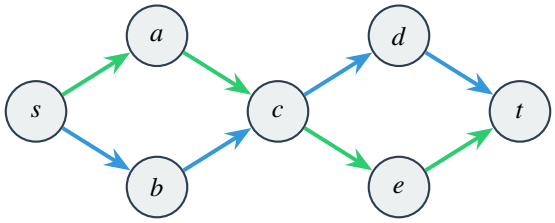
A standard MFDW-SC formulation with binary edge-path variables $x_{uvi} \in \{0, 1\}$ and integer path weights w_i would introduce bilinear products $x_{uvi} \cdot w_i$ in the flow decomposition constraints. Since the weight set W is finite, we index path slots by their weights and use combined variables that record edge use and weight assignment simultaneously, which keeps the model linear.



(a) Instance with required subpath $R_1 = (a, c, e)$ marked in orange. Each edge carries flow 4.



(b) The two paths $s \to a \to c \to d \to t$ and $s \to b \to c \to e \to t$, each of weight 4, decompose the flow but the decomposition violates R_1.



(c) The two paths $s \to a \to c \to e \to t$ and $s \to b \to c \to d \to t$, each of weight 4, decompose the flow and the decomposition satisfies R_1; this is the unique MFDW-SC solution.

Figure 6: An MFDW-SC instance with $W = \{2, 4\}$ and required subpath $R_1 = (a, c, e)$. Without R_1 the flow admits the two distinct 2-path decompositions shown in (b) and (c). The constraint R_1 rules out (b) and selects (c) as the unique MFDW-SC solution.

For a given path cap k, we introduce three families of binary decision variables. For each path slot $i \in [k]$, edge $(u, v) \in E$ and weight index $\ell \in [|W|]$, the indicator

$$z_{uvi\ell} = \begin{cases} 1 & \text{if path } i \text{ uses edge } (u, v) \text{ and has weight } w_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

records the joint choice of edge and weight on a single path slot. For each path slot $i \in [k]$ and weight index $\ell \in [|W|]$, the indicator

$$y_{i\ell} = \begin{cases} 1 & \text{if path } i \text{ is active and has weight } w_\ell, \\ 0 & \text{otherwise,} \end{cases}$$

marks the active weight of slot i. Finally, for each path slot $i \in [k]$ and subpath constraint $j \in [m]$, the indicator

$$r_{ij} = \begin{cases} 1 & \text{if path } i \text{ satisfies constraint } R_j, \\ 0 & \text{otherwise,} \end{cases}$$

records which slot covers which constraint. The formulation therefore uses $k\,|E|\,|W|$ variables of type $z_{uvi\ell}$, $k\,|W|$ variables of type $y_{i\ell}$ and km variables of type r_{ij}. Figure 7 shows how these variables encode a single active path on a small example.

Remark. We sometimes write $z_{ei\ell}$ as shorthand for $z_{uvi\ell}$ where $e = (u, v)$. Similarly, $f(e)$ denotes $f(u, v)$.

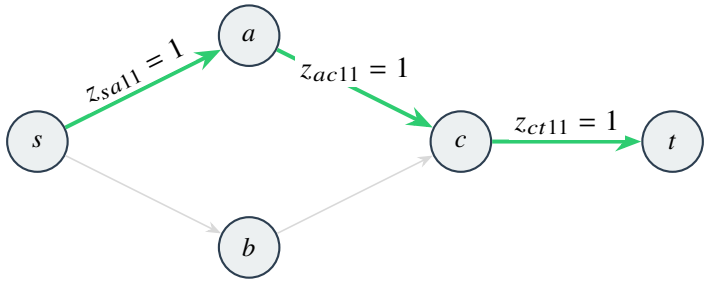


Figure 7: Illustration of the combined variables for one active path. Path slot $i = 1$ uses the highlighted edges with weight index $\ell = 1$, so the corresponding variables $z_{uvi\ell}$ are set to one and $y_{11} = 1$. The faded edges show the rest of the underlying graph.

The combined variable $z_{uvi\ell}$ makes the flow decomposition constraint linear. Since each w_ℓ is a constant from the input, the weighted superposition of paths can be written as

$$\sum_{i=1}^{k} \sum_{\ell=1}^{|W|} z_{uvi\ell} \cdot w_\ell = f(u, v), \quad \forall (u, v) \in E. \tag{4}$$

For each path slot i and weight ℓ, the variables $z_{uvi\ell}$ must form a valid s-t path if $y_{i\ell} = 1$, and must be zero otherwise. This is enforced by the source, sink and conservation constraints (Figure 8)

$$\sum_{(s,v)\in E} z_{svi\ell} = y_{i\ell}, \quad \forall i \in [k], \ell \in [|W|], \tag{5}$$

$$\sum_{(u,t)\in E} z_{uti\ell} = y_{i\ell}, \quad \forall i \in [k], \ell \in [|W|], \tag{6}$$

$$\sum_{(u,v)\in E} z_{uvi\ell} - \sum_{(v,u')\in E} z_{vu'i\ell} = 0, \quad \forall v \in V \setminus \{s, t\}, \ \forall i \in [k], \ell \in [|W|]. \tag{7}$$

Each path slot can be assigned at most one weight from W, or remain unused:

$$\sum_{\ell=1}^{|W|} y_{i\ell} \leq 1, \quad \forall i \in [k]. \tag{8}$$

Each constraint R_j must appear as a subpath in at least one solution path. The coverage constraint is

$$\sum_{i=1}^{k} r_{ij} \geq 1, \quad \forall j \in [m]. \tag{9}$$

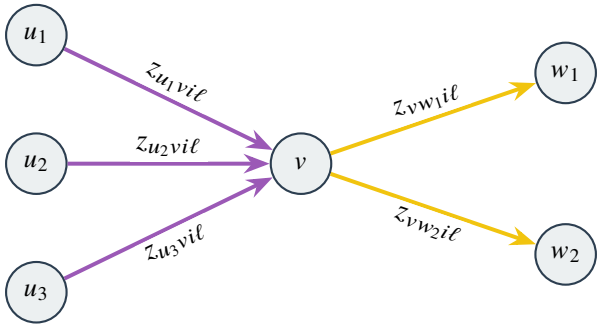


Figure 8: Flow conservation at an interior vertex v for fixed path slot i and weight index ℓ. The three incoming edge indicators (left) must sum to the same value as the two outgoing ones (right). This is the per-slot, per-weight analogue of standard flow conservation and forces $z_{uvi\ell}$ to trace out a single s-t path when $y_{i\ell} = 1$.

Path i can satisfy R_j only if it is active:

$$r_{ij} \leq \sum_{\ell=1}^{|W|} y_{i\ell}, \quad \forall i \in [k], j \in [m], \tag{10}$$

and if it satisfies R_j, it must use every edge of that constraint:

$$\sum_{\ell=1}^{|W|} z_{ei\ell} \geq r_{ij}, \quad \forall i \in [k], j \in [m], e \in R_j. \tag{11}$$

To eliminate symmetric solutions, we optionally impose a non-increasing order on the path weights:

$$\sum_{\ell=1}^{|W|} w_\ell \cdot y_{i\ell} \geq \sum_{\ell=1}^{|W|} w_\ell \cdot y_{(i+1)\ell}, \quad \forall i \in [k-1]. \tag{12}$$

Figure 9 summarizes the complete feasibility formulation. The first feasible k in the outer search is the optimum, and Section 5 explains how the starting lower bound k_0 is computed.

MFDW-SC ILP *Feasibility problem, minimize k by outer iteration*

Variables: $z_{uvi\ell} \in \{0,1\}$ for $(u,v) \in E$, $i \in [k]$, $\ell \in [|W|]$; $y_{i\ell} \in \{0,1\}$ for $i \in [k]$, $\ell \in [|W|]$; $r_{ij} \in \{0,1\}$ for $i \in [k]$, $j \in [m]$.

Objective: Minimize the number of active path slots, $\min \sum_{i=1}^{k} \sum_{\ell=1}^{|W|} y_{i\ell}$.

$$\sum_{i=1}^{k} \sum_{\ell=1}^{|W|} w_\ell \cdot z_{uvi\ell} = f(u,v) \qquad \forall (u,v) \in E \qquad \text{(Flow decomp.)}$$

$$\sum_{v:(s,v)\in E} z_{svi\ell} = y_{i\ell} \qquad \forall i \in [k], \ell \in [|W|] \qquad \text{(Source)}$$

$$\sum_{u:(u,t)\in E} z_{uti\ell} = y_{i\ell} \qquad \forall i \in [k], \ell \in [|W|] \qquad \text{(Sink)}$$

$$\sum_{u:(u,v)\in E} z_{uvi\ell} - \sum_{u':(v,u')\in E} z_{vu'i\ell} = 0 \qquad \forall v \in V \setminus \{s,t\},\ i \in [k],\ \ell \in [|W|]$$

(Conservation)

$$\sum_{\ell=1}^{|W|} y_{i\ell} \le 1 \qquad \forall i \in [k] \qquad \text{(Single weight)}$$

$$z_{uvi\ell} \le y_{i\ell} \qquad \forall (u,v) \in E,\ i \in [k],\ \ell \in [|W|]$$

(Edge activation)

$$\sum_{u:(u,v)\in E} z_{uvi\ell} \le 1 \qquad \forall v \in V \setminus \{s,t\},\ i \in [k],\ \ell \in [|W|]$$

(In-degree)

$$\sum_{u':(v,u')\in E} z_{vu'i\ell} \le 1 \qquad \forall v \in V \setminus \{s,t\},\ i \in [k],\ \ell \in [|W|]$$

(Out-degree)

$$\sum_{i=1}^{k} r_{ij} \ge 1 \qquad \forall j \in [m] \qquad \text{(Coverage)}$$

$$r_{ij} \le \sum_{\ell=1}^{|W|} y_{i\ell} \qquad \forall i \in [k],\ j \in [m]$$

(Active path link)

$$\sum_{\ell=1}^{|W|} z_{ei\ell} \ge r_{ij} \qquad \forall i \in [k],\ j \in [m],\ e \in R_j$$

(Edge link)

$$\sum_{\ell=1}^{|W|} w_\ell \cdot y_{i\ell} \ge \sum_{\ell=1}^{|W|} w_\ell \cdot y_{(i+1)\ell} \qquad \forall i \in [k-1] \qquad \text{(Sym. breaking)}$$

Figure 9: Complete ILP formulation for the MFDW-SC feasibility problem.

Theorem 3.1 below establishes that the ILP correctly models the MFDW-SC problem.

Theorem 3.1 (ILP correctness)**.** *The ILP formulation presented in this section has a feasible solution with parameter k if and only if there exists a MFDW-SC solution with at most k paths.*

Proof. Suppose first that the ILP has a feasible solution. For each path slot $i \in \{1, \dots, k\}$ with $\sum_\ell y_{i\ell} = 1$, let ℓ_i be the unique index such that $y_{i\ell_i} = 1$. By constraints (5), (6) and (7), the variables $\{z_{uvi\ell_i} : (u, v) \in E\}$ define a valid s-t path P_i. Let $\tilde{w}_i = w_{\ell_i}$ denote its weight. Since $\tilde{w}_i \in W$, the set $\mathcal{S} = \{(P_i, \tilde{w}_i) : i \text{ active}\}$ has cardinality at most k.

Constraint (4) guarantees that for every edge (u, v),

$$\sum_{i=1}^{k} \sum_{\ell=1}^{|W|} z_{uvi\ell} \cdot w_\ell = f(u, v).$$

Because each active path slot has exactly one selected weight, this is exactly the required flow decomposition condition

$$\sum_{i \text{ active},\ (u,v) \in P_i} \tilde{w}_i = f(u, v).$$

For the subpath constraints, Equation (9) ensures that for every R_j there exists some slot $i^\star$ with $r_{i^\star j} = 1$. Equation (10) then implies that $i^\star$ is active, and Equation (11) implies that every edge $e \in R_j$ satisfies $z_{ei^\star\ell_{i^\star}} = 1$. Hence $R_j \subseteq P_{i^\star}$, so every constraint is covered. Therefore $\mathcal{S}$ is a valid MFDW-SC solution with at most k paths.

Conversely, suppose $\mathcal{S} = \{(P_1, \tilde{w}_1), \dots, (P_{k'}, \tilde{w}_{k'})\}$ is a valid MFDW-SC solution with $k' \le k$, where each $\tilde{w}_i \in W$. For each $i \in \{1, \dots, k'\}$, let ℓ_i be the index such that $w_{\ell_i} = \tilde{w}_i$. We set $y_{i\ell_i} = 1$ and $y_{i\ell} = 0$ for $\ell \neq \ell_i$, set $z_{uvi\ell_i} = 1$ exactly on the edges of P_i, and set $r_{ij} = 1$ exactly when $R_j \subseteq P_i$. For all remaining path slots $i > k'$, we set every variable to zero.

This assignment satisfies the ILP. Equation (4) holds because $\mathcal{S}$ is already a valid flow decomposition. Equations (5)–(7) hold because each active slot encodes an s-t path. Equation (8) holds because each active slot is assigned exactly one weight and inactive slots are unused. Equation (9) holds because $\mathcal{S}$ covers every constraint. Finally, Equations (10) and (11) hold by construction, since $r_{ij} = 1$ only when path P_i is active and contains every edge of R_j. Therefore the constructed variable assignment is feasible. □

We compare the proposed model with three alternatives. The standard MFD formulation of Dias et al. [3] uses separate variables $x_{uvi} \in \{0, 1\}$ and $w_i \in \mathbb{Z}_{>0}$, which requires linearization of the products $x_{uvi} \cdot w_i$. This introduces $O(k \cdot |E|)$ auxiliary variables and three additional constraints per product term. Our model uses $O(k \cdot |E| \cdot |W|)$ variables, but it avoids that linearization overhead; when $|W|$ is moderate, the two models are comparable in size.

A second possibility is to enumerate all s-t paths and use one variable per path. This explicit path-based view is also used by combinatorial methods such as Toboggan [7], but the number of s-t paths can be exponential, so such formulations are only practical on very small graphs or within a more elaborate column-generation framework.

A third alternative appears in the MFDW setting without subpath constraints. Grigorjew et al. [4] use a weight-indexed formulation with variables $x_{uv\ell} \in \mathbb{N}$ that count paths of weight w_ℓ on edge (u, v). Their formulation is very compact, with $O(|E| \cdot |W|)$ variables, but does not distinguish individual paths and therefore cannot enforce subpath coverage. Our formulation is larger by a factor of k, and that extra structure is precisely what lets us encode the constraints from $\mathcal{R}$.

The MFDW-SC problem generalizes both MFDW and MFDSC. The MFDW problem [4] is the special case with an empty constraint set, that is, $\mathcal{R} = \emptyset$. In our ILP this removes the variables r_{ij} and the covering constraints in Equations (9)–(11). The MFDSC problem [16] is the special case with an unrestricted weight set, which can be represented by taking $W = \{1, 2, \ldots, \max_{e \in E} f(e)\}$. Our formulation captures both restrictions in a single model.

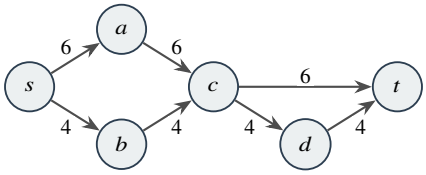
Table 1 compares the asymptotic complexity of our formulation with the most relevant prior work. The main increase in complexity comes from the $O(k \cdot |V| \cdot |W|)$ path conservation constraints in Equation (7), since path validity must be enforced for every path slot i and every possible weight w_ℓ. In practice, this cost is offset by starting the outer iteration from a lower bound k_0 rather than from $k = 1$, as described in Section 5.

Table 1: Comparison of ILP formulations for flow decomposition variants

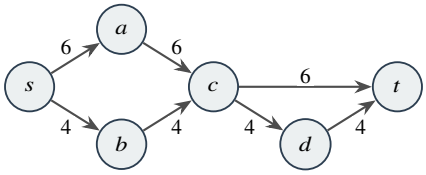
Work	Variables	Constraints	Problem
Dias et al. [3]	$O(k\|E\|)$	$O(k\|E\|)$	MFD
Williams et al. [16]	$O(k\|E\|+km)$	$O(k(\|E\|+m))$	MFDSC
Grigorjew et al. [4]	$O(\|E\|\|W\|)$	$O(\|V\|\|W\|+\|E\|)$	MFDW
This work	$\mathbf{O(k(\|E\|\|W\|+m))}$	$\mathbf{O(k(\|V\|\|W\|+m)+\|E\|)}$	**MFDW-SC**

The formulation used throughout this thesis can represent subpath constraints, unlike the weight-indexed approach of Grigorjew et al. [4]. Its number of variables and constraints is polynomial in the input parameters k, $|E|$, $|W|$ and m, unlike a path-enumeration formulation, and it remains fully linear. This makes it suitable for the solver developed in the later chapters.

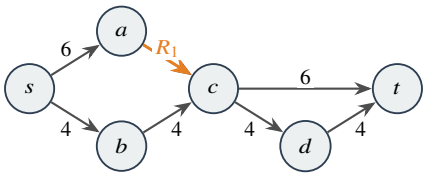
Figure 10 compares the four problems on the same network to make the effect of each restriction explicit. The unrestricted MFD solution uses a weight-6 path $s \to a \to c \to t$ and a weight-4 path $s \to b \to c \to d \to t$. Restricting the weights to $W = \{2, 4\}$ splits the weight-6 route into weights 4 and 2, while the single subpath constraint $R_1 = (a, c)$ alone is already covered by the original weight-6 route.



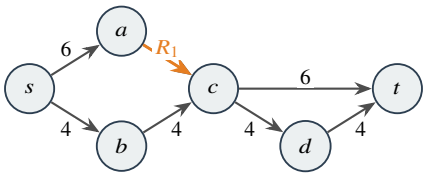
(a) MFD: no restrictions, $k^\star = 2$.



(b) MFDW: $W = \{2, 4\}$, $k^\star = 3$.



(c) MFDSC: $R_1 = (a, c)$, $k^\star = 2$.



(d) MFDW-SC: both restrictions, $k^\star = 3$.

Figure 10: The four flow decomposition variants on the same six-node network. Weight restrictions increase the optimum from two to three paths; the single subpath constraint alone does not.

3.2 Computational complexity

MFDW-SC inherits NP-hardness from MFDSC.

Theorem 3.2 (MFDW-SC is NP-complete)**.** *The decision version of MFDW-SC ("Does there exist a solution with $\leq k$ paths?") is NP-complete.*

Proof. We prove NP-hardness by reduction from MFDSC, which was shown NP-hard by Williams et al. [16].

Membership in NP: Given a candidate solution $(P_1, w_1), \ldots, (P_k, w_k)$, we can verify in polynomial time that each P_i is a valid s-t path ($O(k \cdot |E|)$), that each $w_i \in W$ ($O(k)$), that the flow decomposition holds ($O(|E| \cdot k)$) and that each R_j is covered ($O(m \cdot k \cdot |E|)$). The total verification cost is $O(k \cdot m \cdot |E|)$, which is polynomial, so MFDW-SC $\in$ NP.

Hardness: We reduce MFDSC to MFDW-SC. Given an MFDSC instance $I = (G, f, \mathcal{R})$, construct an MFDW-SC instance $I' = (G, f, W, \mathcal{R})$ with

$$W = \{1, 2, \ldots, F_{\max}\}, \quad \text{where } F_{\max} = \max_{e \in E} f(e).$$

The set W is built in $O(F_{\max})$ time.

Claim: I has a solution with $\leq k$ paths if and only if I' has a solution with $\leq k$ paths.

($\Rightarrow$) Any MFDSC solution $(P_1, w_1), \ldots, (P_k, w_k)$ has $w_i \in \{1, \ldots, F_{\max}\}$ because $w_i \leq f(e)$ for every $e \in P_i$ and f takes values in $\{1, \ldots, F_{\max}\}$. Hence $w_i \in W$ and the solution is feasible for I'.

($\Leftarrow$) Any MFDW-SC solution for I' is feasible for I as well: dropping the weight restriction $w_i \in W$ leaves the same paths, weights, flow decomposition and subpath coverage.

Since MFDSC is NP-hard, MFDW-SC is NP-hard. Together with NP membership, MFDW-SC is NP-complete. □

Worst-case complexity is therefore not changed by the weight restriction, although restricting $|W| \ll \max_{e \in E} f(e)$ can reduce the search space in practice.

4 Structural theory

This chapter develops the structural framework that the rest of the thesis builds on. A path of weight w can only use edges (u, v) with $f(u, v) \geq w$, so the subgraph G_w of edges that pass this threshold depends only on w. Distinct weights in W that yield the same G_w share the same set of s-t paths, the same reachability structure and therefore the same answer to every question about which subpath constraints can be routed through a path of that weight. They can be analyzed together.

To see why this matters, suppose we want to determine for each subpath constraint R_j and each weight $w \in W$ whether there exists an s-t path of weight w that contains R_j as a subpath. A naive approach tests every weight individually with a separate reachability search on G_w, costing $O(|V| + |E|)$ per check and $O(|W| \cdot m \cdot (|V| + |E|))$ in total. Grouping weights by their layer graph reduces this to one check per group, with the group's answer copied to all weights it contains. The remainder of the chapter formalizes this grouping, derives a per-edge integer-decomposition number $\pi(e)$ from it and combines π with the antichain argument of Cáceres et al. [1] into a lower bound on the size of any feasible MFDW-SC decomposition.

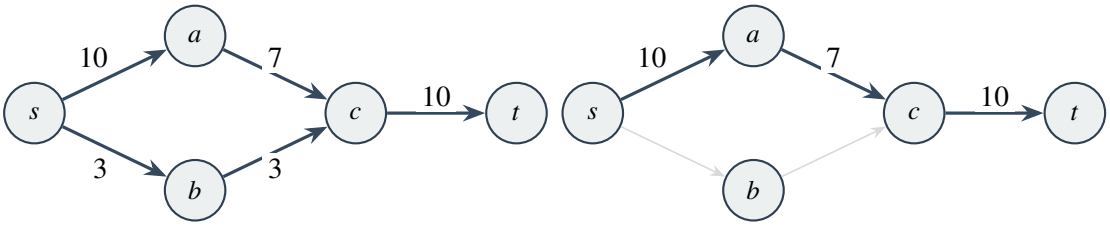
4.1 Topological and feasible weight sets

The first object we need is the subgraph of edges that can carry a given weight. Definition 4.1 below isolates this subgraph.

Definition 4.1 (Weight layer graph)**.** Given a flow network $G = (V, E, f)$ and a threshold $w \in \mathbb{Z}_{>0}$, the *weight-w layer graph* is defined as:

$$G_w := (V, E_w), \quad \text{where} \quad E_w := \{\, e \in E \mid f(e) \geq w \,\}.$$

Remark. Intuitively, G_w retains only edges that can support flow of at least w units. Any path in a flow decomposition having weight w must be entirely contained within G_w.



(a) G_3: All edges present ($E_3 = E$) **(b)** G_7: Only edges with $f(e) \geq 7$

Figure 11: Layer graph monotonicity. As the weight threshold increases from $w = 3$ to $w = 7$, edges with insufficient flow are removed (shown gray). The path $s \to b \to c \to t$ exists in G_3 but not in G_7. This illustrates $E_7 \subsetneq E_3$; more generally, Lemma 4.1 gives $E_{w_2} \subseteq E_{w_1}$ whenever $w_2 > w_1$.

Lemma 4.1 (Layer monotonicity)**.** *If $w_2 > w_1$, then $E_{w_2} \subseteq E_{w_1}$, and thus G_{w_2} is a subgraph of G_{w_1}.*

Proof. If $e \in E_{w_2}$, then by definition $f(e) \geq w_2 > w_1$, so $f(e) \geq w_1$, thus $e \in E_{w_1}$. □

As the weight threshold increases, the layer graph becomes smaller, as illustrated in Figure 11. For weights exceeding the largest edge flow, the layer graph contains no edges.

4.2 Weight equivalence classes

Layer monotonicity implies that as w grows the edge set E_w shrinks one step at a time, only when w crosses one of the distinct flow values present in E. Allowed weights that lie between consecutive flow values therefore induce identical layer graphs and behave identically in every reachability question we will ask. Grouping the weights in W into equivalence classes lets us perform each topological check once per class rather than once per allowed weight.

Definition 4.2 (Weight equivalence and weight classes)**.** Let $G = (V, E, f)$ be a flow network, let $W \subset \mathbb{Z}_{>0}$ be the finite allowed weight set and let $T = \{t_1 < t_2 < \cdots < t_q\}$ be the set of distinct flow values appearing in E, ordered increasingly. For weights $w, w' \in W$ write

$$w \sim w' \quad \Longleftrightarrow \quad E_w = E_{w'},$$

so $\sim$ is the equivalence relation on W given by equality of layer graphs. The *representative* of w is

$$[w] := \begin{cases} \min\{\, t \in T \mid t \geq w \,\} & \text{if such } t \text{ exists,} \\ +\infty & \text{otherwise,} \end{cases}$$

and the *weight class* indexed by $r \in T \cup \{+\infty\}$ is

$$C_r := \{\, w \in W \mid [w] = r \,\}.$$

We use the convention $E_{+\infty} = \emptyset$ and $G_{+\infty} = (V, \emptyset)$.

Proposition 4.2 (Partition property)**.** *The weight classes* $\{C_r : r \in T \cup \{+\infty\}\}$ *partition* W*, with at most* $\min\{|W|, |T| + 1\} \leq |E| + 1$ *non-empty classes.*

Proof. Every $w \in W$ has a unique representative $[w] = \min\{t \in T \cup \{+\infty\} : t \geq w\}$, so the classes C_r partition W. Since there can be no more non-empty classes than allowed weights or representatives, and since $|T| \leq |E|$, the number of non-empty classes is at most $\min\{|W|, |T| + 1\} \leq |E| + 1$. □

Proposition 4.3 (Layer invariance)**.** *For any weight* $w \in C_r$*, we have* $E_w = E_r$ *(where* $E_r := \{\, e \in E \mid f(e) \geq r \,\}$ *for* $r \in T$ *and* $E_{+\infty} := \emptyset$*) and thus* $G_w = G_r$*. Consequently, any topology-only predicate* Π *(such as reachability) satisfies* $\Pi(G_w) = \Pi(G_r)$ *for all* $w \in C_r$*.*

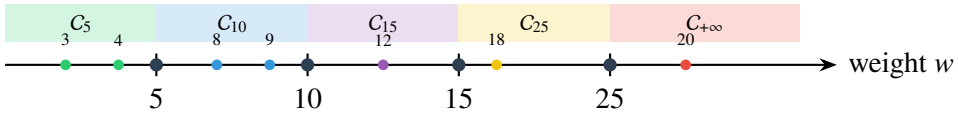


Figure 12: Weight class partitioning based on flow values $T = \{5, 10, 15, 25\}$. Weights in the same colored region share the same representative and thus induce identical layer graphs. For example, the sample weights $w = 8$ and $w = 9$ both have representative $[w] = 10$, so $G_8 = G_9 = G_{10}$.

Proof. Let $w \in C_r$, so $[w] = r$. If $r = +\infty$, then w is larger than every flow value, and both E_w and $E_{+\infty}$ are empty. Otherwise, we show $E_w = E_r$. First, let $e \in E_w$, so $f(e) \geq w$. Since $r = [w] = \min\{t \in T : t \geq w\}$, we have $r \geq w$. But every flow value $f(e)$ belongs to T, and because $f(e) \geq w$ while r is the smallest value in T that is at least w, it follows that $f(e) \geq r$. Thus $e \in E_r$. Conversely, let $e \in E_r$, so $f(e) \geq r$. By definition of the representative, $r \geq w$, and hence $f(e) \geq r \geq w$. Thus $e \in E_w$.

Therefore, $E_w = E_r$, which means $G_w = G_r$. Since any topology predicate Π (such as "is there a path from s to t?") depends only on the edge set, we have $\Pi(G_w) = \Pi(G_r)$. □

This is the main result of the section: all weights in a class C_r induce the same layer graph G_r, so we only need to compute topology once per representative (see Figure 12 for an example partition), not once per weight w.

4.3 Topological satisfiability sets

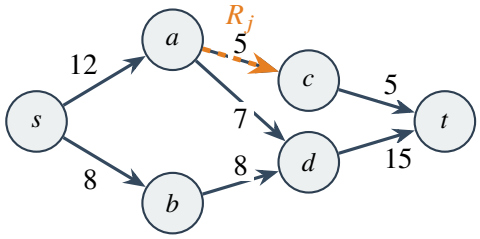
For each subpath constraint R_j, only some weights $w \in W$ allow R_j to lie on a feasible s-t path: the path must stay inside the layer graph G_w, which only contains edges with $f(e) \geq w$. The set of such admissible weights is what the rest of the section formalizes.

Definition 4.3 (Topological satisfiability set)**.** Given a subpath constraint R_j and a weight $w \in W$, let the *weight-w layer graph* be G_w. Define

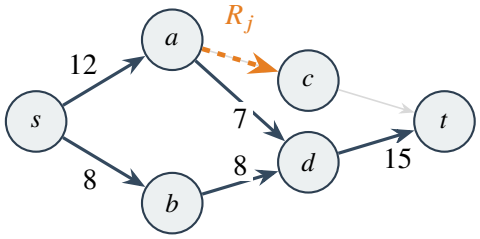
$$W_{\text{topo}}(R_j) \;:=\; \big\{\, w \in W \mid \exists \text{ an } s\text{-}t \text{ path in } G_w \text{ that contains } R_j \,\big\}.$$

The set $W_{\text{topo}}(R_j)$ contains exactly those weights w for which the layer graph G_w retains enough structure to route an s-t path through R_j. Writing $R_j = (e_1, \ldots, e_p)$ with $e_1 = (a_0, a_1)$ and $e_p = (a_{p-1}, a_p)$, this requires that G_w contains every edge of R_j, that s can reach the first vertex a_0 in G_w and that the last vertex a_p can reach t in G_w. Figure 13 illustrates these checks on a small layer graph.

Let T be the set of distinct edge flows $\{f(e) \mid e \in E\}$, ordered increasingly, and let C_r denote the class of weights whose layer equals G_r (see Proposition 4.3). Since all $w \in C_r$ share the same layer G_r, we can compute W_{topo} *per class* and lift the result to all weights in that class.



(a) Original graph, constraint $R_j = (a, c)$



(b) Layer G_6: constraint edge removed

Weight w	$R_j \subseteq E_w$	$s \rightsquigarrow a$ in G_w	$c \rightsquigarrow t$ in G_w
$w = 4$	Yes	Yes	Yes
$w = 6$	No	Yes	No
$w = 10$	No	Yes	No

Figure 13: Topological satisfiability for constraint $R_j = (a, c)$ with $f(a, c) = 5$. In layer G_6, edge (a, c) is removed since $5 < 6$, making vertex c unreachable from s. Thus $w = 4 \in W_{\text{topo}}(R_j)$ but $w = 6, 10 \notin W_{\text{topo}}(R_j)$. A weight is topologically satisfiable only when all three conditions hold simultaneously.

Corollary 4.4 (Classwise computation of W_{topo})**.** *Let $\Pi_{R_j}(G_r)$ be the predicate "there exists an s-t path in G_r that contains R_j." Then*

$$W_{topo}(R_j) = \bigcup_{\substack{r \in T \cup \{+\infty\} \\ \Pi_{R_j}(G_r)=\text{true}}} C_r.$$

Proof. By Proposition 4.3, every $w \in C_r$ induces the same layer graph $G_w = G_r$, hence $\Pi_{R_j}(G_w) = \Pi_{R_j}(G_r)$. The set $W_{\text{topo}}(R_j)$ is therefore the union of those classes for which the predicate holds. □

To evaluate whether a constraint $R_j = (e_1, \ldots, e_p)$ is satisfiable in a given layer graph G_r, with $e_1 = (a_0, a_1)$ and $e_p = (a_{p-1}, a_p)$, we check three conditions in sequence: first, that all edges of R_j are present in E_r; second, that the source s can reach the first vertex a_0 of R_j in G_r via BFS or DFS; and third, that the last vertex a_p can reach the sink t in G_r. Each check runs in $O(|V| + |E|)$ time.

If all three checks pass, we set $\Pi_{R_j}(G_r) = \text{true}$ and add the entire class C_r to $W_{\text{topo}}(R_j)$, since all weights in the class share the same layer graph by Proposition 4.3.[1] The pseudocode is Algorithm 5 in Appendix B.

The classwise computation of $W_{\text{topo}}(R_j)$ has two direct consequences for the ILP. First, combined with the edge bottleneck and reachability tests, it supports variable

[1] By Lemma 4.1, $W_{\text{topo}}(R_j) \cap W$ is downward closed, so one could binary-search over class representatives. We test each class instead because $|T|$ is small in practice (between 1 and 48 on our benchmark), and a binary search over already-grouped weights does not justify the implementation complexity.

pruning by identifying edge–weight combinations that cannot appear in any feasible path. Second, it enables stronger linking constraints: the covering variable r_{ij} can be restricted to activate only when path i uses a weight from $W_{\text{valid}}(R_j)$, tightening the LP relaxation. Both effects reduce the size of the model and the branch-and-bound search.

4.4 Bottlenecks and feasible weight sets

The edge bottleneck gives an equivalent arithmetic view of the same restriction: a path of weight w can use only edges e with $f(e) \geq w$, so w is bounded by the smallest edge flow along the subpath.

Definition 4.4 (Bottleneck and feasible weight set)**.** For $R_j = (e_1, \ldots, e_p)$, the *bottleneck weight* is $U_{R_j} := \min_{e \in R_j} f(e)$, the largest weight that can traverse every edge of R_j. The *feasible weight set* is

$$W_{\text{valid}}(R_j) \;:=\; W_{\text{topo}}(R_j) \;\cap\; \left\{ w \in \mathbb{Z}_{>0} \;\middle|\; w \leq U_{R_j} \right\}.$$

For a single edge $e = (u, v)$ we similarly write $W_{\text{valid}}(e) = W_{\text{valid}}(u, v)$ for the weights $w \in W$ with $w \leq f(u, v)$, $u \in \mathcal{S}_{[w]}$ and $v \in \mathcal{T}_{[w]}$.

A weight w lies in $W_{\text{valid}}(R_j)$ precisely when two conditions hold simultaneously. First, w must be topologically admissible: the layer graph $G_w = (V, E_w)$ must contain all edges of R_j, a path from s to the first vertex a_0 of R_j and a path from the last vertex a_p of R_j to t. Equivalently, G_w must contain an s-t path that visits R_j in order. Second, w must satisfy the bottleneck constraint $w \leq U_{R_j}$, so that w does not exceed the smallest edge flow along R_j. This second condition is implied by the first when W_{topo} is computed exactly; we keep it explicit because the implementation applies the bottleneck as a cheap filter after classwise reachability.

Applying the bottleneck bound U_{R_j} costs $O(|R_j|)$ and can remove high weights early in implementations that cache topological reachability by class. This shrinks the candidate weight set linked to r_{ij} and yields *weight forcing* when $W_{\text{valid}}(R_j)$ is a singleton.

4.5 Weighted antichain lower bound

The layered model of the previous sections defines, for every edge e, a feasible weight set $W_{\text{valid}}(e)$ that respects both the edge capacity and the layer-graph reachability conditions. We use that set to derive a per-edge integer-decomposition number $\pi(e)$, and combine it with the antichain argument of Cáceres et al. [1] into a lower bound on the size of any feasible MFDW-SC decomposition.

Definition 4.5 (Edge integer-decomposition number)**.** For an edge $e = (u, v) \in E$ with feasible weight set $W_{\text{valid}}(u, v)$, let

$$\pi(e) := \min \left\{ \sum_{w \in W_{\text{valid}}(u,v)} x_w \;:\; \sum_{w \in W_{\text{valid}}(u,v)} x_w \, w = f(e),\; x_w \in \mathbb{Z}_{\geq 0} \right\},$$

with $\pi(e) := +\infty$ if no integer combination exists.

The value $\pi(e)$ is the minimum number of paths of weights from $W_{\text{valid}}(e)$ whose weights sum to $f(e)$. Figure 14 contrasts this bound against the unweighted antichain bound on a small example. It is computed by a coin-change dynamic program in $O(f(e) \cdot |W_{\text{valid}}(u, v)|)$ time and is well defined whenever the instance is feasible. An edge with $\pi(e) = +\infty$ admits no decomposition and the instance can be rejected before solving.

Theorem 4.5 (Weighted antichain lower bound)**.** *Let $\mathcal{A} \subseteq E$ be an edge antichain in G. Every feasible MFDW-SC decomposition uses at least*

$$k_0^{\text{anti}}(\mathcal{A}) := \sum_{e \in \mathcal{A}} \pi(e)$$

paths.

Proof. Let $\mathcal{S} = \{(P_1, \tilde{w}_1), \ldots, (P_k, \tilde{w}_k)\}$ be any feasible decomposition. For each $e \in \mathcal{A}$, let $I_e := \{i : e \in P_i\}$ denote the set of paths covering e.

For every $i \in I_e$, the weight $\tilde{w}_i$ lies in $W_{\text{valid}}(e)$. Indeed $\tilde{w}_i \le f(e)$ by capacity (flow conservation on edge e gives $\sum_{j : e \in P_j} \tilde{w}_j = f(e)$ with positive summands, so each individual $\tilde{w}_i$ is at most $f(e)$); the same capacity argument applied along every edge of P_i via flow conservation extends this to $P_i \subseteq G_{\tilde{w}_i}$. By Proposition 4.3 the endpoints of e therefore lie in $\mathcal{S}_{[\tilde{w}_i]}$ and $\mathcal{T}_{[\tilde{w}_i]}$, so $\tilde{w}_i$ satisfies the layer-reachability conditions of $W_{\text{valid}}(e)$.

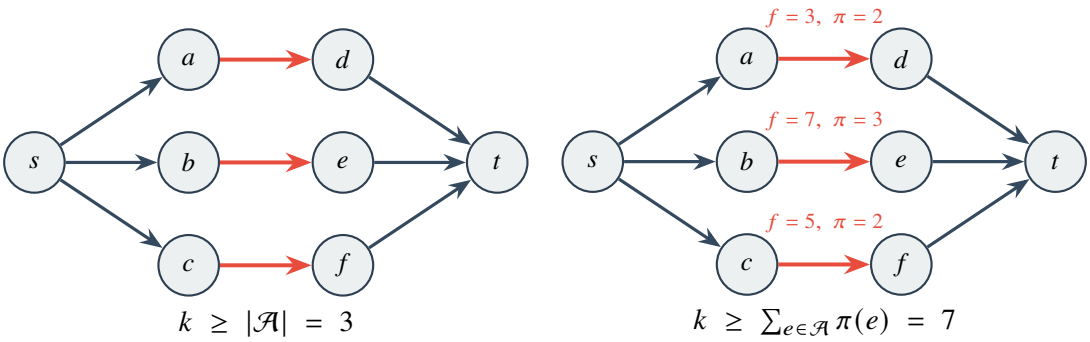
Flow conservation at e gives $\sum_{i \in I_e} \tilde{w}_i = f(e)$. Setting $x_w := |\{i \in I_e : \tilde{w}_i = w\}|$ produces a feasible integer decomposition of $f(e)$ over $W_{\text{valid}}(e)$ with $|I_e|$ summands, so $|I_e| \ge \pi(e)$.

Because $\mathcal{A}$ is an antichain, the sets $\{I_e : e \in \mathcal{A}\}$ are pairwise disjoint: no s-t path can contain two distinct antichain edges. Therefore

$$k = \sum_{i=1}^{k} 1 \;\ge\; \sum_{e \in \mathcal{A}} |I_e| \;\ge\; \sum_{e \in \mathcal{A}} \pi(e). \qquad \square$$

The bound $k_0^{\text{anti}}(\mathcal{A})$ tightens the unweighted antichain bound of Cáceres et al. [1]. Their bound counts antichain edges; this bound counts the integer covers each antichain edge requires. The per-edge ceiling $\lceil f(e)/\max W_{\text{valid}}(e) \rceil$ uses only the largest feasible weight; $\pi(e)$ uses the entire feasible set. Both improvements can be strict. For example, with $f(e) = 100$ and $W_{\text{valid}}(e) = \{7, 13\}$, the ceiling gives $\lceil 100/13 \rceil = 8$, while the integer decomposition requires $\pi(e) = 10$, achieved by $5 \cdot 13 + 5 \cdot 7 = 100$.

To turn Theorem 4.5 into a single number we maximize over edge antichains. Computing a maximum-weight antichain on a DAG with edge weights $\pi(e)$ is polynomial: it reduces to a minimum-cost flow on a derived bipartite construction, the standard weighted analogue of Dilworth's theorem [1]. The implementation computes $\pi(e)$ for every edge during the lower-bound calculation in Algorithm 16; in practice, it evaluates a greedy collection of high-weight antichains rather than solving the exact



Unweighted antichain bound [1]: each of the three antichain edges (red) lies on a different s-t path, giving $k \geq 3$.

Weighted antichain bound (Theorem 4.5) with $W = \{1, 2, 4\}$: each edge needs $\pi(e)$ paths to carry its flow (3=2+1, 7=4+2+1, 5=4+1), and the antichain disjointness lets the path counts add to 7.

Figure 14: Unweighted antichain bound (left) versus the weighted antichain bound of Theorem 4.5 (right) with $W = \{1, 2, 4\}$. Replacing each unit count by the per-edge cost $\pi(e)$ raises the bound from 3 to 7 on the same antichain.

maximum-weight antichain problem. This heuristic already dominates the unweighted antichain bound on almost every instance of our benchmark, and every antichain it returns still gives a valid lower bound by Theorem 4.5.

Corollary 4.6 (Three-term starting bound)**.** *Combining Theorem 4.5 with the source bound and a constraint-incompatibility bound, every feasible MFDW-SC decomposition uses at least*

$$k_0 = \max\left\{ \left\lceil \frac{F_s}{\max W} \right\rceil,\ \max_{\mathcal{A}\ \text{antichain}} \sum_{e \in \mathcal{A}} \pi(e),\ \ \omega(\mathcal{G}_{\text{incomp}}) \right\}$$

paths, where F_s is the total source outflow and $\omega(\mathcal{G}_{\text{incomp}})$ is the size of the largest clique in the constraint-incompatibility graph $\mathcal{G}_{\text{incomp}}$ whose nodes are the subpath constraints and whose edges connect pairs of constraints that cannot be satisfied by the same s-t path under any feasible weight (constructed in Section 4.6).

Proof. Let $\mathcal{S} = \{(P_1, \tilde{w}_1), \ldots, (P_k, \tilde{w}_k)\}$ be a feasible decomposition. Summing the flow conservation equation across all outgoing edges from s gives $\sum_{i=1}^{k} \tilde{w}_i = F_s$, and since each $\tilde{w}_i \leq \max W$ we obtain $k \geq \lceil F_s / \max W \rceil$, which is the first term. The second term is the statement of Theorem 4.5 maximized over edge antichains. For the third term, every pair of constraints joined by an edge in $\mathcal{G}_{\text{incomp}}$ must be covered by distinct paths by construction of the graph, so a clique of size $\omega(\mathcal{G}_{\text{incomp}})$ forces that many distinct paths. The maximum of the three terms is therefore a valid lower bound on k. □

The first term bounds the global flow balance, the second term bounds the per-edge integer decomposability and the third term bounds the constraint coverage. Each term

is tight on a different family of instances. The implementation combines the three terms in Algorithm 16 and uses the resulting k_0 to start the iterative search at k_0 rather than at $k = 1$, as described in Section 5.3. The empirical evaluation in Section 6 reports which of the three terms binds in practice on the present benchmark.

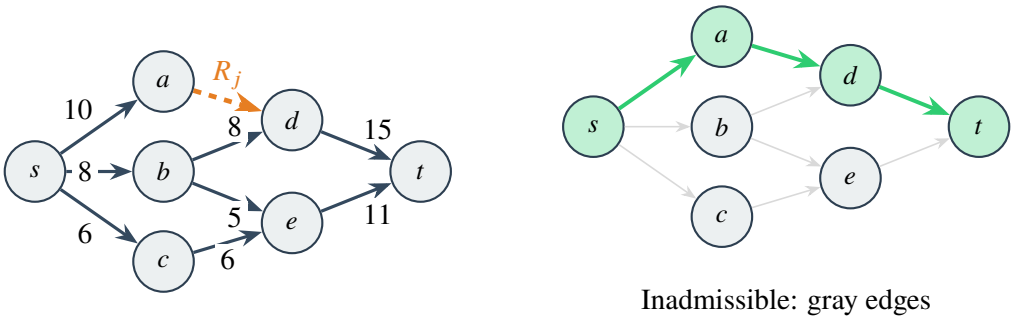
4.6 Further valid inequalities

The layered model also suggests several classes of valid inequalities that tighten the LP relaxation without changing the integer optimum. Their practical effect is reported as a separate ablation in Section 6.

The first class is the *edge conflict cuts*. When a path satisfies constraint R_j using weight w_ℓ, its topology is restricted: the path must lie entirely within the layer graph $G_{[w_\ell]}$ (containing only edges with flow at least $[w_\ell]$) and must traverse all edges of R_j as a subsequence. These restrictions mean that many edges in E cannot appear in any path satisfying R_j at weight w_ℓ. For example, if $R_j = (a, b, c)$ and $w_\ell = 10$, then any edge (x, y) with $f(x, y) = 5 < 10$ is absent from the layer graph $G_{[w_\ell]}$ and therefore cannot co-occur with R_j in a path of weight w_ℓ.

Definition 4.6 (Admissible edge set)**.** Fix a constraint $R_j = (e_1, \ldots, e_p)$ and a representative $r \in T$. Let $G_r = (V, E_r)$ be the layer graph and write $R_j \trianglelefteq P$ when R_j appears in order along path P. Define the admissible edge set at layer r by

$$\mathrm{Adm}(R_j, r) := \left\{ (u, v) \in E_r \ : \ \exists\ s\text{-}t \text{ path } P \text{ in } G_r \text{ with } R_j \trianglelefteq P \text{ and } (u, v) \in P \right\}.$$



(a) Network with constraint

(b) $\mathrm{Adm}(R_j, r)$ at $r = 7$

Figure 15: Admissible edge set $\mathrm{Adm}(R_j, r)$ at layer $r = 7$ for the constraint $R_j = (a, d)$. The left panel shows the original layer graph G_7 with the constraint edges in orange; the right panel highlights in green the edges that can co-occur with R_j on some s-t path in G_7. The conflict cuts of Section 4.6 forbid each non-admissible edge from carrying a path that also covers R_j at weight class r, by enforcing $z_{uvi\ell} \le 1 - r_{ij}$ whenever $(u, v) \notin \mathrm{Adm}(R_j, [w_\ell])$.

Remark. If $(u, v) \notin \mathrm{Adm}(R_j, r)$, then no s-t path in G_r can simultaneously contain R_j and (u, v).

Figure 15 illustrates which edges co-occur with R_j on some s-t path at a given threshold. For each constraint R_j, path i, weight index ℓ and edge $(u, v) \notin \mathrm{Adm}(R_j, [w_\ell])$, we add the conflict cut

$$z_{uvi\ell} \le 1 - r_{ij}. \tag{14}$$

If path i satisfies constraint R_j (so $r_{ij} = 1$), then path i cannot use inadmissible edge (u, v) at weight w_ℓ (so $z_{uvi\ell} = 0$). These cuts tighten the LP relaxation by eliminating fractional solutions that would violate topological feasibility.

Adding conflict cuts for all (R_j, w_ℓ, u, v) tuples can increase the model size substantially. On instances with many paths, constraints and weight classes, the number of ECC constraints can reach into the thousands, enlarging the LP relaxation that must be solved at every node of the branch-and-bound tree. Two strategies can mitigate this cost. The first is to limit which constraints receive ECC cuts, for instance by adding them only for constraints with at most τ feasible weights where the cuts are most likely to be binding; this is what Algorithm 11 in Appendix B does, with τ defaulting to 10 in our implementation. The second is to avoid adding the cuts upfront altogether and instead separate them lazily during the solve. The model is built without any ECC constraints and a solver callback inspects each integer-feasible candidate to check for violations, adding only the violated constraints as lazy cuts. We describe this lazy separation strategy in Section 4.6 and report its empirical effect in Section 6.

A second class is the *forced single-path constraints*. Some constraints are fully determined by their feasible weight set and the layer-graph topology. Suppose $W_{\text{valid}}(R_j) = \{w^\star\}$ is a singleton and the layer graph $G_{[w^\star]}$ contains a unique s-t path that traverses R_j. Then any feasible decomposition must contain that path with weight $w^\star$, since no other path of any feasible weight can cover R_j. The path itself is found by walking forward from $s(R_j)$ towards t and backward from $t(R_j)$ towards s in $G_{[w^\star]}$, accepting only vertices whose successor or predecessor is unique. The path can then be fixed before the ILP is built by Algorithm 12 in Appendix B. The observation strictly generalizes the *safe paths* of Khan et al. [6], which are subpaths that appear in every flow decomposition regardless of its size and are described in detail in Section 5.1: every safe path of G is a unique-path region in $G_{[w^\star]}$ for the bottleneck weight, but not every forced subpath constraint of this kind is a safe path of the original graph. The empirical effect of fixing these forced paths is reported in Section 6.

A third class is the *multiplicity upper bounds*. For each constraint R_j, we derive an upper bound on the number of paths that can simultaneously satisfy it. Let $U_{R_j} = \min_{(u,v)\in R_j} f(u, v)$ denote the bottleneck flow along the edges of R_j, and let $w^j_{\min} = \min_{w\in W_{\text{valid}}(R_j)} w$ be the smallest feasible weight for R_j. Since each path satisfying R_j carries at least $w^j_{\min}$ units of flow through every edge of R_j, at most $M_j = \lfloor U_{R_j}/w^j_{\min} \rfloor$ paths can do so without exceeding the bottleneck capacity. We add the constraint

$$\sum_{i=1}^{k} r_{ij} \le M_j$$

whenever $M_j < k$. This is a valid inequality that tightens the LP relaxation by preventing the solver from distributing fractional coverage across too many paths. The pseudocode is Algorithm 13 in Appendix B.

A fourth class is the *edge integer-decomposition cuts*. For an edge (u, v), write $\pi(u, v) := \pi((u, v))$ for the per-edge integer-decomposition number from Definition 4.5. This number produces a valid inequality at the path-slot level. Every feasible

decomposition routes at least $\pi(u, v)$ paths through edge (u, v), so for each edge with $\pi(u, v) > \lceil f(u, v)/\max W_{\text{valid}}(u, v) \rceil$ we add

$$\sum_{i=1}^{k} \sum_{\ell \,:\, w_\ell \in W_{\text{valid}}(u,v)} z_{uvi\ell} \;\geq\; \pi(u, v).$$

We refer to these as edge integer-decomposition cuts (EDC). They differ from the edge conflict cuts of Section 4.6 in two ways. The conflict cuts forbid a single (z, r) pair when an edge cannot co-occur with a covered constraint, while the decomposition cuts impose a global lower bound on the number of z activations on each edge. The two families address complementary structure and are both included in the ablation in Section 6. We add an EDC only when $\pi(u, v)$ exceeds the per-edge ceiling $\lceil f(u, v)/\max W_{\text{valid}}(u, v) \rceil$, since otherwise the LP relaxation already implies the bound. The pseudocode is Algorithm 14 in Appendix B.

A fifth class is the *constraint incompatibility cuts*. When multiple subpath constraints are present, certain pairs of constraints cannot be satisfied by the same path under any feasible weight. Two constraints R_{j_1} and R_{j_2} are incompatible if, for every weight class $r \in T$ such that both constraints have all their edges present in G_r, there is no s-t path in G_r that traverses both R_{j_1} and R_{j_2} (in either order). We detect such pairs during the structural analysis by checking, for each weight class, whether the endpoint of one constraint can reach the start of the other within the corresponding layer graph. The pairs form the edges of the constraint-incompatibility graph $\mathcal{G}_{\text{incomp}}$ used in the third term of the starting lower bound (15).

For each incompatible pair (j_1, j_2) and each path index i, we add the constraint

$$r_{ij_1} + r_{ij_2} \leq 1$$

which prevents any single path from being assigned to cover both constraints simultaneously. These cuts are valid because no feasible path can satisfy both constraints and they tighten the LP relaxation by eliminating fractional solutions that would split coverage across incompatible constraints. The graph $\mathcal{G}_{\text{incomp}}$ is constructed during structural analysis by Algorithm 7; Algorithm 15 then adds the corresponding cuts when this optional component is enabled.

A practical refinement of the edge conflict cuts is *lazy separation*. The edge conflict cuts (ECC) of Section 4.6 forbid a path from using an inadmissible edge when it satisfies a particular constraint at a particular weight. While these cuts are valid inequalities that can tighten the LP relaxation, adding all of them upfront can introduce thousands of constraints into the model. For example, on an instance with 23 paths, 3 constraints and 10 weight classes, the number of ECC constraints can exceed 12 000. This overhead increases the size of the LP relaxation solved at every node of the branch-and-bound tree, and on instances where few or none of these constraints are ever active, the cost can outweigh the benefit.

To address this, we adopt a lazy separation strategy: instead of adding all ECC constraints when the model is built, we add none of them initially and rely on a solver callback to detect violations during the branch-and-bound process. Whenever the

solver finds an integer-feasible candidate solution, the callback checks whether any path in the candidate uses an inadmissible edge for its assigned constraint and weight class. If a violation is found (that is, if $z_{uvi\ell} = 1$ and $r_{ij} = 1$ but $(u, v) \notin \mathrm{Adm}(R_j, [w_\ell])$), the corresponding constraint $z_{uvi\ell} \leq 1 - r_{ij}$ is added to the model as a lazy cut and the solver continues.

This has the same theoretical guarantees as adding all ECC constraints upfront: every feasible integer solution satisfies all ECC constraints, and any infeasible candidate is rejected by the callback. The practical advantage is that the LP relaxation remains small throughout the solve, and only genuinely violated constraints are ever added. In our experiments, the variable-pruning preprocessing already eliminates most of the edge-weight combinations that ECC would target, so the callback rarely needs to add any cuts at all. The lazy separation strategy thus avoids the overhead of thousands of redundant constraints while preserving correctness.

4.7 Summary and algorithmic implications

Table 2: Structural framework components and the role they play in the solver of Section 5.

Component	Definition	Role
C_r	Weight equivalence classes	Compress $\lvert W\rvert$ queries to $\lvert T\rvert$
G_r	Layer graphs	Encode topology per weight class
$\Pi_{R_j}(G_r)$	Path existence predicate	Test constraint feasibility
$W_{\text{valid}}(R_j)$	Feasible weight set	Drive variable pruning and linking
$\pi(e)$	Per-edge integer-decomposition number	Weighted antichain lower bound
$\mathrm{Adm}(R_j, r)$	Admissible edges	Generate edge conflict cuts

The framework reduces runtime in two ways. First, layer invariance and feasibility analysis remove impossible weight-edge combinations before the ILP is built, which is the principal source of model reduction. On many real instances this eliminates well over half of the $z_{uvi\ell}$ variables, and a substantially larger fraction on the most structured ones. Second, the per-edge integer-decomposition number $\pi(e)$ yields the weighted antichain lower bound of Theorem 4.5, which lets the iterative search start at a value of k that is often equal to the optimum. The additional valid inequalities developed in Section 4.6 (edge conflict cuts, forced single-path constraints) tighten the formulation further, but their practical effect is decided empirically in Section 6.

These structural insights are the foundation of the solver implementation described in Section 5.

5 Layered ILP solver

This chapter turns the structural theory of Section 4 into a working solver. The solver takes a flow network $G = (V, E, f)$, a weight set W and subpath constraints $\mathcal{R}$, and returns an optimal MFDW-SC decomposition through a four-phase pipeline. Phase 1 reduces the graph by transformations that preserve feasible decompositions. Phase 2 carries out the structural analysis: weight equivalence classes, layer graphs and feasible weight sets. Phase 3 builds a refined ILP that uses these results to remove provably infeasible variables and to add the refinements derived in Section 4. Phase 4 solves fixed-k models for increasing k starting from the bound of Corollary 4.6, stopping at the first feasible value. Appendix B gives pseudocode for the implemented routines.

5.1 Phase 1: Graph preprocessing

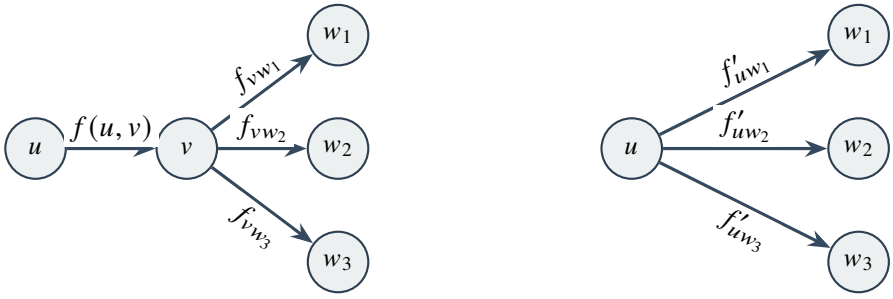
Before constructing the ILP, we apply two graph transformations that reduce the problem size without changing the optimal solution. Both techniques are standard in the flow decomposition literature and can be applied regardless of the formulation used.

The first transformation is the Y-to-V reduction of Kloster et al. [7]. This operation removes a non-protected vertex when one side of it has degree 1. If v has a unique predecessor u, every path entering v must enter through (u, v), so each outgoing edge (v, w) can be replaced by a shortcut edge (u, w) with flow $f(v, w)$. Symmetrically, if v has a unique successor w, each incoming edge (u, v) can be replaced by a shortcut edge (u, w) with flow $f(u, v)$. The contraction is applied iteratively until no more contractible vertices remain, producing a reduced graph $G' = (V', E')$ with $|V'| \leq |V|$ and $|E'| \leq |E|$. Vertices that lie on an edge of some subpath constraint $R_j \in \mathcal{R}$ are protected from contraction so that every constraint can still be expressed in G'; this protected set is the role of $V_{\text{protected}}$ in Algorithm 1.

This contraction preserves feasible decompositions bijectively: any path through v maps to the corresponding shortcut edge, and every shortcut edge stores the original two-edge segment in σ so it can be expanded after solving. When the ILP returns a solution on G', each contracted edge is expanded back through σ, recovering the corresponding paths in G with no loss of information. In terms of model size, the reduction decreases the number of $z_{uvi\ell}$ variables from $O(k \cdot |E| \cdot |W|)$ to $O(k \cdot |E'| \cdot |W|)$ and similarly reduces the number of flow conservation constraints. Figure 16 illustrates the contraction for a vertex with in-degree 1.

The second transformation augments the constraint set with edge sequences that must appear together in every flow decomposition, using the excess flow criterion of Khan et al. [6]. At a vertex v, if the incoming edge (u, v) has flow f and only one outgoing edge (v, w) also carries flow f, then flow conservation forces any decomposition path using (u, v) to continue through (v, w). Extending this reasoning over chains of such forced continuations yields safe paths: subpaths that appear in every valid decomposition regardless of its size.

We add these safe paths to the constraint set, forming $\mathcal{R}_{\text{all}} := \mathcal{R}_{\text{original}} \cup \mathcal{R}_{\text{safe}}$. Since safe paths are present in every decomposition by definition, this does not change the



(a) Before reduction. **(b)** After contraction of v.

Figure 16: Y-to-V reduction: contracting a node v with in-degree 1. The flows are preserved: $f'(u, w_i) = f(v, w_i)$ for each i.

feasible region of the ILP. It provides the solver with additional structural information. Each safe path added to $\mathcal{R}_{\text{all}}$ introduces covering and linking constraints in the ILP, tightening the LP relaxation and often allowing the solver to fix variables early in the branch-and-bound process.

5.2 Phase 2: Structural analysis

The second phase computes the weight class structure and feasible weight sets, applying the theory from Section 4 to the preprocessed graph.

We begin by computing the set T of distinct flow values appearing on edges of G', and partitioning W into weight classes based on which layer graph each weight induces (Definition 4.2). Two weights w, w' belong to the same class if and only if $[w] = [w']$, where $[w]$ is the smallest flow value in T that is at least w; if $[w] = t_a$, then $t_{a-1} < w \leq t_a$ with the convention $t_0 = 0$. For each representative $r \in T \cup \{+\infty\}$, we construct the layer graph $G_r = (V, E_r)$, using $E_r = \{e \in E : f(e) \geq r\}$ for finite r and $E_{+\infty} = \emptyset$. We then precompute, for each layer graph, the set $\mathcal{S}_r$ of vertices reachable from s and the set $\mathcal{T}_r$ of vertices that can reach t. These reachability sets are computed once per class representative via breadth-first search, requiring $O((|T| + 1) \cdot (|V| + |E|))$ time in total.

For each constraint R_j and each edge (u, v), we compute the feasible weight sets $W_{\text{valid}}(R_j)$ and $W_{\text{valid}}(u, v)$ as defined in Definition 4.4, using the classwise computation from Corollary 4.4. Rather than testing each weight individually, we test one representative per equivalence class and extend the result to all weights in that class. This reduces the cost from $O(m \cdot |W| \cdot (|V| + |E|))$ to $O(m \cdot |T| \cdot (|V| + |E|))$, which is a saving whenever $|T| \ll |W|$. The same structural pass constructs the constraint-incompatibility graph $\mathcal{G}_{\text{incomp}}$ used by the lower bound and, optionally, by the incompatibility cuts.

5.3 Phase 3: Refined ILP construction

The results of the structural analysis feed into a refined version of the ILP from Section 3. We first describe the core refinements: model changes that follow directly

from the framework and remain enabled in every configuration reported in Section 6. A separate group of valid inequalities and a lazy separation routine, whose practical effect is studied in the ablation of Section 6, is then summarized as the *optional inequalities* at the end of the subsection.

The first refinement prunes variables. The baseline ILP creates a variable $z_{uvi\ell}$ for every combination of edge (u, v), path index i and weight index ℓ. Many of these variables correspond to assignments that are topologically impossible: a path of weight w_ℓ cannot traverse an edge (u, v) if w_ℓ exceeds the edge flow, or if no s-t path of that weight class can reach (u, v). For each edge (u, v) we collect the weights that pass these basic tests into the per-edge feasible set

$$W_{\text{valid}}(u, v) = \{\, w \in W : w \le f(u, v),\ u \in \mathcal{S}_{[w]},\ v \in \mathcal{T}_{[w]} \,\}.$$

The variable $z_{uvi\ell}$ is created only when $w_\ell \in W_{\text{valid}}(u, v)$. The first condition respects the edge capacity, the second ensures that the corresponding layer graph can reach u from s and the third ensures that t can be reached from v. If one of these tests fails, the variable is omitted. Any feasible solution would set it to zero anyway, so this does not exclude any valid decomposition. On instances with several distinct flow levels ($|T| > 1$), this pruning can eliminate a substantial fraction of the z variables, with reductions of up to 81% on individual instances depending on the graph structure and the ratio $|W|/|T|$. The corresponding procedure is Algorithm 8 in Appendix B.

The second refinement is a tightening of the cover variables. For each constraint R_j, the structural analysis produces a feasible weight set $W_{\text{valid}}(R_j) \subseteq W$ consisting of those weights under which R_j can be satisfied by some s-t path. We tighten the formulation by restricting the covering variable r_{ij} to activate only when path i uses a weight from this feasible set:

$$r_{ij} \le \sum_{\ell\,:\,w_\ell \in W_{\text{valid}}(R_j)} y_{i\ell}$$

In the special case where $W_{\text{valid}}(R_j) = \{w^\star\}$ is a singleton with index $\ell^\star$, this simplifies to $r_{ij} \le y_{i\ell^\star}$, which forces any path satisfying R_j to have weight $w^\star$. Weight forcing can reduce the search space when the structural analysis determines that a constraint is compatible with only one or two weights. The pseudocode for adding the linking constraints is Algorithm 9 in Appendix B.

Two further lightweight refinements come from the same analysis. Without any ordering on the paths, any permutation of a feasible solution is also feasible, leading the solver to explore up to $k!$ equivalent solutions. We break this symmetry by imposing a non-increasing weight ordering on the paths, as specified in Equation (12) and added by Algorithm 10 in Appendix B; this eliminates symmetric permutations without excluding any distinct solution. The structural analysis also yields a branching priority for the solver: constraints with a small feasible weight set $|W_{\text{valid}}(R_j)|$ are the most restricted covering decisions, so we set the Gurobi branching priority of each r_{ij} to a value inversely proportional to $|W_{\text{valid}}(R_j)|$, which is a one-line change with no effect on the model size.

Before beginning the iterative search, we compute a lower bound k_0 on the number of paths needed. The implementation uses the source term from Corollary 4.6, the incompatibility-clique term and a computed weighted-antichain term $\widehat{k}_{\text{anti}}$:

$$k_0 = \max\left\{\left\lceil \frac{F_s}{\max W} \right\rceil,\ \widehat{k}_{\text{anti}},\ \omega(\mathcal{G}_{\text{incomp}})\right\} \tag{15}$$

where $F_s = \sum_{(s,v)\in E} f(s,v)$ is the total source outflow, $\widehat{k}_{\text{anti}} = \sum_{e\in\widehat{\mathcal{A}}} \pi(e)$ for the antichain $\widehat{\mathcal{A}}$ returned by Algorithm 16, and $\omega(\mathcal{G}_{\text{incomp}})$ is the size of the largest clique in the constraint-incompatibility graph computed during structural analysis.

The first term is a global flow bound: even if every path carries the largest available weight, at least $\lceil F_s/\max W\rceil$ paths are needed. The second term is a weighted-antichain bound from Theorem 4.5: each antichain edge contributes the minimum number of paths required to integer-decompose its flow over the feasible weight set, and antichain disjointness lets these counts add. Using a heuristic antichain may make $\widehat{k}_{\text{anti}}$ smaller than the exact maximum in Corollary 4.6, but it remains a valid lower bound. The third term is a constraint-coverage bound: any pair of constraints in an incompatibility clique must lie on different paths, so the largest such clique is itself a lower bound on k.

Starting the iteration at k_0 rather than at $k = 1$ avoids solving infeasible ILPs for small values of k. The pseudocode that combines the three terms appears in Algorithm 16.

An optional group of valid inequalities may be added on top of the core refinements. The group adds the additional valid inequalities derived in Section 4.6: edge conflict cuts (here added by lazy separation), forced single-path constraints, constraint multiplicity upper bounds, edge integer-decomposition cuts and constraint incompatibility cuts. They are correct and follow from the structural framework, but their effect on solve time depends on the instance and on which other components are active. Each is reported separately in the ablation of Section 6, where we record whether enabling the component reduces or increases total runtime relative to the core refinements alone.

5.4 Phase 4: Iterative solving

The solver searches for the minimum k by building and solving the refined ILP for $k = k_0, k_0 + 1, \ldots$. Each value of k receives the full time budget. If the ILP solver proves infeasibility for some k, we know that $k^\star > k$ and move on to $k + 1$. If the solver solves the model to optimality for some k, that value is optimal since all smaller values have already been ruled out. If the run reaches the time limit or another stopping condition first, the instance is treated as not converged. The full loop is Algorithm 17 in Appendix B; the decomposition is recovered from the integer-feasible Gurobi solution by Algorithm 18, which reads the active $z_{uvi\ell}$ and $y_{i\ell}$ variables and reconstructs the $k^\star$ paths and their weights.

To accelerate the search, we provide the ILP solver with a warm start from a greedy decomposition heuristic. The greedy algorithm repeatedly extracts a shortest s-t path in the residual graph and assigns it the largest feasible weight. The

resulting decomposition is usually not optimal, but it gives the solver a useful starting point. The implementation also bounds the search range from above by $k_{\max} = \min\{\text{max\_k\_cap}, \max(k_0, k_{\text{struct}}, k_{\text{greedy}}, k_{\text{flowUB}})\}$, where $k_{\text{struct}} = |E(G')| - |V(G')| + 2$ is the Vatinlen upper bound [14], k_{greedy} is the number of paths returned by the greedy warm start, $k_{\text{flowUB}} = \lceil F_s / \min W \rceil$ is the worst-case flow bound and max_k_cap is a user-configurable safety cap (set to a value well above the largest expected $k^\star$). The iteration stops when k reaches $k_{\max}$ if no feasible model has been found by then.

The implementation also exposes a single-solve mode in which the path cap is fixed once from the greedy decomposition and a single larger ILP minimizes the number of active path slots inside that cap. Section 6 reports both modes for the baseline and Layered formulations; the optional-inequality ablation is restricted to iterative mode, since that is the only mode in which all formulations were run.

5.5 Pipeline summary

The four phases form a pipeline that progressively refines the problem before handing it to the ILP solver. Graph preprocessing reduces the size of the input. Structural analysis identifies which weight-edge-constraint combinations are feasible. The refined ILP construction translates these feasibility results into a smaller model with tighter relaxations through the core refinements, and the optional inequalities are added on top when requested. The iterative solver then searches for the minimum k starting from the structural lower bound k_0. Figure 17 visualizes the data flow between the phases, and Algorithm 19 in Appendix B gives the end-to-end pseudocode, chaining the per-phase algorithms cited above into a single call from the input $(G, f, W, \mathcal{R})$ to a decomposition.

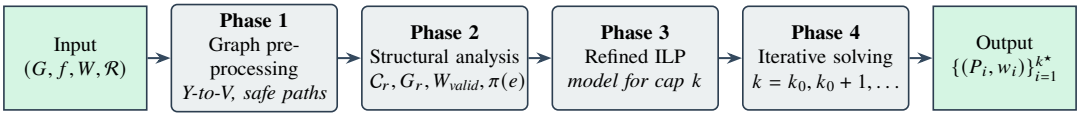


Figure 17: Solver pipeline; the per-phase algorithms are composed by Algorithm 19 in Appendix B.

For the experiments, we compare this pipeline against a baseline solver that follows the same iterative structure but does not perform any structural analysis. The baseline creates all $z_{uvi\ell}$ variables without pruning, does not compute weight equivalence classes or layer graphs and does not add the linking constraints, branching priority or starting lower bound from the core refinements. The optional inequalities are switched on independently in the ablation. Both solvers share the same symmetry-breaking constraints, so the observed differences come entirely from the model reduction.

6 Experimental results

This chapter reports the results of running the Layered ILP solver of Section 5 and the baseline ILP of Section 3 on real splice graphs from RNA sequencing data. Both solvers return the same minimum $k^\star$, so the experiments compare two solving strategies for the same problem. Ratios between configurations are reported as geometric means.

6.1 Setup and data

The benchmark contains 325 splice-graph instances from real RNA sequencing data. Each instance represents one gene; vertices correspond to exons and edges carry normalized read counts. Graphs range from 50 to 319 vertices, and the number of distinct flow values, which determines the number of weight equivalence classes, ranges from one to 48.

For every instance we used the same weight set

$$W = \{2^0, 2^1, \ldots, 2^{\lfloor \log_2 F_{\max} \rfloor}\},$$

where $F_{\max}$ is the largest edge flow on the instance. The powers-of-two choice guarantees that every positive integer flow is decomposable and keeps W small enough for exact ILP solving. We generated three subpath constraints per instance, each consisting of two to four consecutive edges, drawn from the ground-truth decomposition where possible. The same instances, weight set and constraints are passed to every configuration.

Each fixed-k ILP receives a time budget of 300 seconds. Runs that do not return an optimal solution within that budget are reported separately and excluded from the per-configuration averages. All configurations share the same Gurobi parameters (Presolve 2, MIPFocus 1, Cuts 2, Heuristics 0.05) and the same greedy warm start as described in Section 5.4.

All experiments were run with Gurobi 13.0 and a Python implementation of the surrounding pipeline. The implementation of both solvers, together with the scripts that reproduce every figure and table in this chapter, is released at `https://github.com/immone/mfdw-sc`.

We report eleven configurations split between two solve modes. In *iterative* mode the solver tests $k = k_0, k_0 + 1, \ldots$ and stops at the first feasible value; in *single-solve* mode it builds the model once with a fixed path-slot count taken from the greedy decomposition and minimizes the number of active slots inside that cap. The suffix `_iter` marks iterative mode and `_sing` marks single-solve mode. The prefix `B` denotes the baseline ILP, `min` the Layered ILP with only the core refinements (variable pruning, linking constraints, branching priority and the antichain starting bound) on top of the Y-to-V graph reduction, the `plus*` prefixes denote the Layered ILP with one optional inequality switched on, and `full` denotes the Layered ILP with all optional inequalities switched on at once. Table 3 below lists every configuration.

Within the time budget, each configuration solves between 311 and 324 instances to optimality; the remaining instances reach the time limit and are recorded as not

Table 3: Configurations evaluated in this chapter. The `min_iter` configuration is the one we recommend on this benchmark; the other configurations enable additional optional inequalities or single-solve mode and are reported for completeness in the ablation.

Identifier	Solver	Mode
`B_iter`	Baseline ILP	iterative
`B_sing`	Baseline ILP	single-solve
`min_iter`	Layered ILP, core refinements only	iterative
`min_sing`	Layered ILP, core refinements only	single-solve
`plusECC_iter`	Layered ILP + edge conflict cuts	iterative
`plusClique_iter`	Layered ILP + constraint incompatibility cuts	iterative
`plusMultUB_iter`	Layered ILP + multiplicity upper bounds	iterative
`plusEDC_iter`	Layered ILP + edge integer-decomposition cuts	iterative
`plusForced_iter`	Layered ILP + forced single-path constraints	iterative
`full_iter`	Layered ILP + all optional inequalities	iterative
`full_sing`	Layered ILP + all optional inequalities	single-solve

converged. Per-configuration optimality counts on the full benchmark are reported in Table 4, including the 14 instances on which at least one configuration timed out. Different configurations time out on different instances, so the only fair cross-configuration comparison is on the intersection of the eleven per-configuration successful sets, which contains 311 instances. Throughout this chapter we therefore restrict the quantitative comparisons (geometric-mean ratios, medians, distributions) to this 311-instance *common set*.

Table 4: OPTIMAL count per configuration out of 325 instances under the time limit. Each row is a marginal count; the 311-instance common set used in the rest of the chapter is the intersection across all eleven configurations. The recommended Layered ILP and the optional-inequality variants other than `plusEDC` and `full` match or improve on the baseline; the single-solve mode variants and any configuration that includes the edge integer-decomposition cuts lose convergence on a few additional instances.

Configuration	OPTIMAL	Time-limit
`B_iter`	319	6
`B_sing`	319	6
`min_iter`	321	4
`min_sing`	316	9
`plusECC_iter`	324	1
`plusClique_iter`	321	4
`plusMultUB_iter`	322	3
`plusEDC_iter`	316	9
`plusForced_iter`	321	4
`full_iter`	318	7
`full_sing`	311	14

6.2 Number of k-iterations until the first feasible ILP

The iterative solver builds and tests a sequence of ILP models, one for each candidate value of k from k_0 upwards, until the first feasible value is reached. Each model that is proved infeasible produces no decomposition, so the tightness of the starting bound k_0 controls how much of the total runtime goes into building and testing infeasible models. Figure 18 shows the per-instance distribution of the number of models tested by each solver. The baseline ILP starts at $k = 1$ and iterates from $k = 1$ up to $k^\star$, taking 7.7 iterations on average and 34 in the worst case. The Layered ILP starts at the bound of Corollary 4.6 and is feasible on the first model for 304 out of 311 instances, with the remaining seven needing a second model. The bound matches $k^\star$ whenever the maximum-weighted antichain identifies a cut of G on which each edge admits no shorter integer decomposition over W than the number of paths actually crossing it (Theorem 4.5); on the seven instances where the bound is off by one, some antichain edge admits a decomposition into fewer summands from W than the number of paths through it.

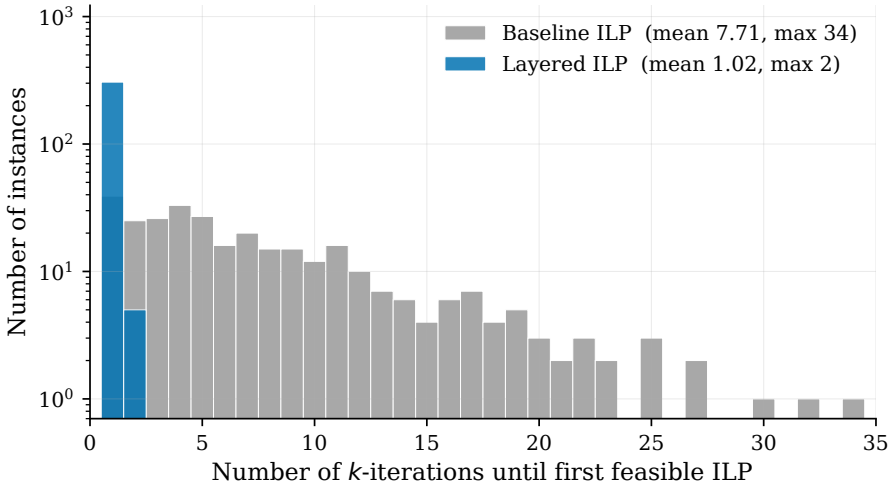


Figure 18: Number of k-iterations until the first feasible ILP for each of the 311 common-set instances.

6.3 Total solve time

Figure 19 shows the distribution of total solve time per instance for the four main configurations: the baseline ILP and the Layered ILP, each in iterative and single-solve mode. The vertical axis is on a log scale because the runtime spans more than four orders of magnitude across the benchmark. The baseline iterative solver has a median of 0.5 seconds; switching it to single-solve mode brings the median down to 0.16 seconds because the wasted small-k models are no longer built. The Layered ILP starts at the antichain bound, so those infeasibility models are already absent, and its iterative median is 0.03 seconds. Single-solve mode does not help the Layered ILP further (median 0.04 seconds): there is no infeasibility loop left to remove, and

the larger single-solve model is slightly more expensive to build. Table 5 reports the head-to-head between the iterative versions: the Layered ILP is faster than the baseline on 310 out of 311 instances, with a geometric mean speedup of 7.5 and a median speedup of 8.6.

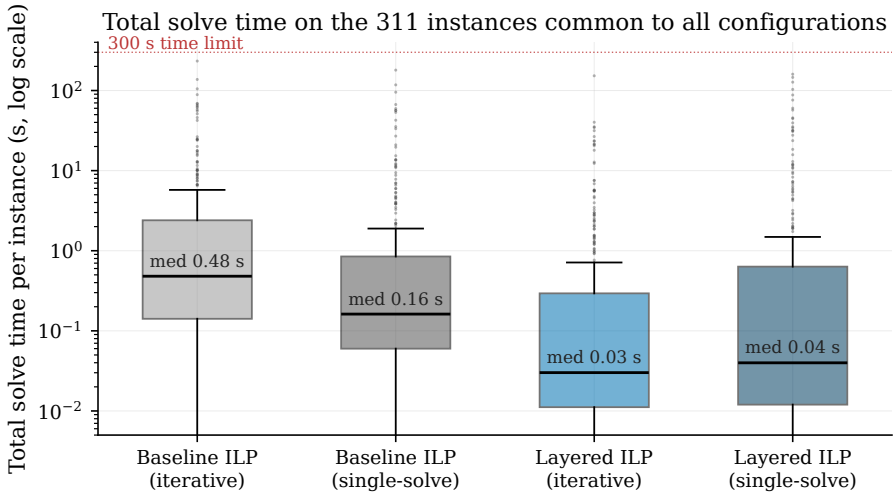


Figure 19: Distribution of total solve time per instance for the four main configurations on the 311 common-set instances. Box widths are quartiles; whiskers extend to 1.5 times the inter-quartile range. The dashed line marks the per-iteration time limit. Single-solve mode helps the baseline by skipping the wasted small-k models.

Table 5: Layered ILP versus baseline ILP, both in iterative mode, on the 311-instance common set. The geometric mean of the speedup ratios r_i is $\exp(\frac{1}{n}\sum_i \log r_i)$.

Metric	**Value**
Common-set instances	311
$k^\star$ agreement	311 / 311 (100%)
Geometric mean speedup	7.51
Median speedup	8.63
Layered ILP faster	310 / 311 (99.7%)
Mean variable reduction	72.4%
Mean baseline k-iterations	7.7
Mean Layered ILP k-iterations	1.02

6.4 Where the Layered ILP spends its time

The total runtime of the Layered ILP splits into three phases: structural preprocessing (computing weight classes, layer graphs, feasible weight sets and the antichain lower bound), building the ILP model in Gurobi and solving the ILP. Figure 20 reports the median time spent on each phase across four difficulty regimes defined by the total Layered ILP runtime. The vertical axis is on a log scale because the ILP solve

time grows by four orders of magnitude across the regimes (from 3 ms to 23 s), while the model build grows by less than two (from 11 ms to 185 ms) and structural preprocessing stays within 0.2 to 2 ms throughout. Preprocessing is therefore at most a few milliseconds and never accounts for a noticeable share of the wall-clock time. The model build dominates only on the easiest regime, where the ILP terminates in milliseconds; on every harder regime the ILP solve takes over. The structural work done outside Gurobi is therefore cheap; the savings come from the variables and constraints it removes from the model that the solver actually handles.

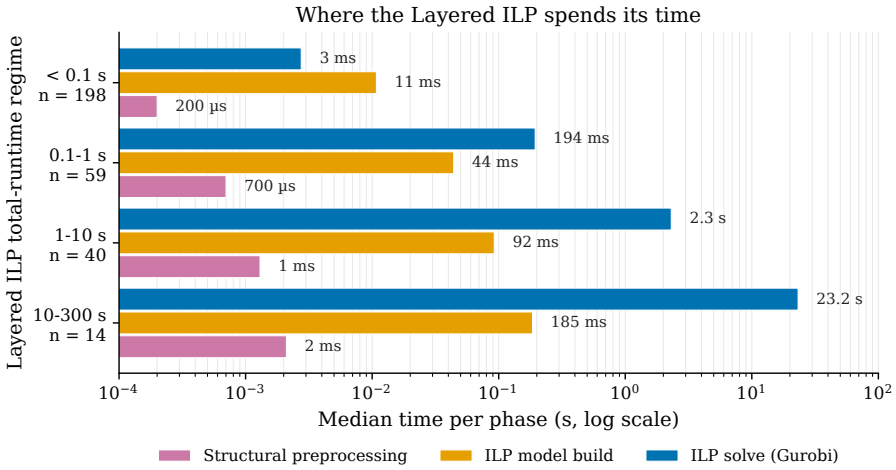


Figure 20: Median per-phase time of the Layered ILP within each difficulty regime, on a log axis. The ILP solve grows by four orders of magnitude across regimes; structural preprocessing stays sub-millisecond throughout.

6.5 Method behavior across difficulty regimes

The aggregate distributions of Figure 19 can hide systematic differences between methods at the easy and hard ends of the benchmark. Figure 21 reports the median runtime of the four main configurations broken down by difficulty regime, where regimes are defined by the iterative baseline's runtime. The Layered ILP iterative configuration leads in every regime. The median speedup over the baseline peaks at 16× on the 0.1 to 1 s regime, where the baseline spends most of its time iterating through small infeasible k values, and decreases to roughly 4× on the 10 to 300 s regime, where the bottleneck shifts from k-iteration count to the size of the final ILP. The plot also shows that the gap between iterative and single-solve mode within the Layered ILP, hardly visible in Figure 19, widens on the hardest regime: the single-solve median is 25.7 s against 6.3 s for the iterative configuration.

6.6 Which lower bound term is binding?

The starting bound of Corollary 4.6 is the maximum of three lower bounds: the source-flow bound, the weighted antichain bound and the constraint-incompatibility bound.

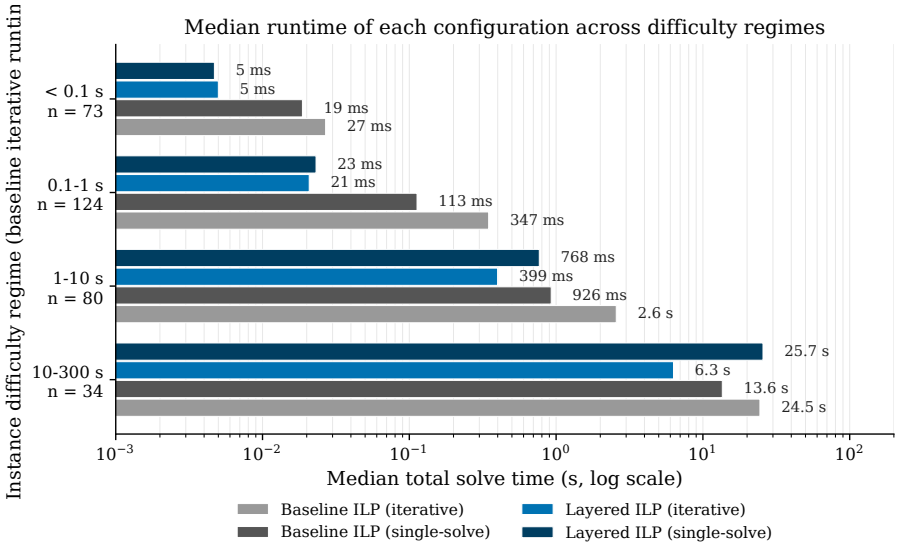


Figure 21: Median total solve time of each of the four main configurations across difficulty regimes defined by the iterative baseline's runtime. The Layered ILP iterative mode dominates in every regime; single-solve mode helps the baseline but begins to hurt the Layered ILP on the hardest regime.

Knowing which of the three is binding identifies the structural argument responsible for the iteration collapse of Figure 18. Figure 22 reports the per-instance binding term: the weighted antichain bound binds on 86.5% of the common-set instances, the source-flow bound on 13.2% and the constraint-incompatibility bound on the remaining 0.3% (a single instance). The weighted antichain bound of Theorem 4.5 is therefore the structural property that makes the iterative search start at the optimum on this benchmark.

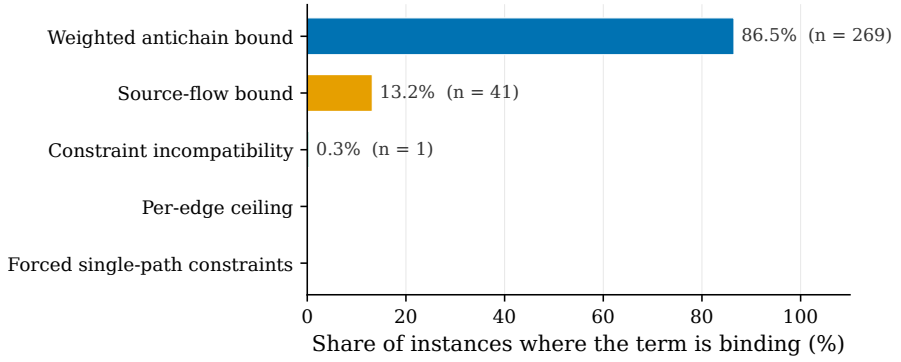


Figure 22: Share of the 311 common-set instances on which each term of Corollary 4.6 is the binding lower bound. The weighted antichain bound is binding on most instances.

6.7 Effect of the optional inequalities

The optional-inequality group adds four valid inequalities (multiplicity upper bounds, edge integer-decomposition cuts, constraint incompatibility cuts and the lazy edge-

conflict callback) and one set of forced single-path constraints derived from safe-path detection. To assess each component, we switch it on individually on top of the recommended Layered ILP and compare per-instance runtime on the common set. Table 6 reports the share of instances on which each addition helps and hurts, together with the geometric-mean runtime ratio; Figure 23 shows the full distribution per component.

Table 6: Effect of each optional-inequality component when switched on individually on top of the recommended Layered ILP, measured on the 311-instance common set. The ratio is computed per instance as runtime with the cut divided by runtime without it, so values below one mean the cut helped on that instance.

Component	Helped (%)	Hurt (%)	Geometric mean ratio
Forced single-path constraints	67.9	26.7	0.99
Multiplicity upper bounds	60.8	30.9	0.99
Constraint incompatibility cuts	48.9	42.1	0.99
Edge-conflict cuts (lazy)	30.5	66.9	1.00
Edge integer-decomposition cuts	29.9	63.3	1.19

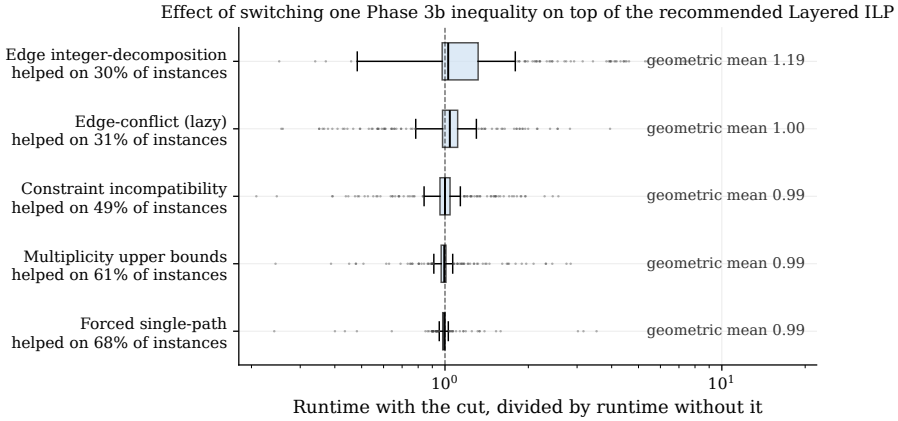


Figure 23: Distribution of the per-instance runtime ratio of each optional-inequality component on top of the recommended Layered ILP. A ratio below one means the added inequality helped on that instance. The geometric mean is annotated on the right.

Three of the five components have a geometric-mean ratio equal to one. They help on a slight majority of instances and slow down a slight minority, so they are theoretically valid but offer no measured gain on this benchmark. Edge-conflict cuts in the lazy callback are similarly close to neutral but lose more often than they win.

Only the edge integer-decomposition cuts are clearly detrimental, and the aggregate ratio of 1.2 in Table 6 understates the situation because the slowdown grows with instance difficulty. Figure 24 splits the geometric-mean ratio by regime: 1.0 on the easiest regime, 1.4 on the 0.1 to 1 s regime, 1.9 on the 1 to 10 s regime and 2.0 on the 10 to 300 s regime, with the share of slowed instances rising from 56% to 79% over the same range. The effect on the configuration that switches all five components on at once is visible in Table 4: `full_iter` converges on three fewer instances than

the recommended configuration and its geometric-mean speedup over the baseline drops from 7.5 to 6.1. This matches the LP-relaxation analysis of Section 4.6: once the core variable pruning is in place, the LP relaxation already implies the per-edge integer-decomposition bound for almost every edge, so the EDC cuts enlarge the constraint matrix without tightening the relaxation, and the larger matrix is what slows the harder instances.

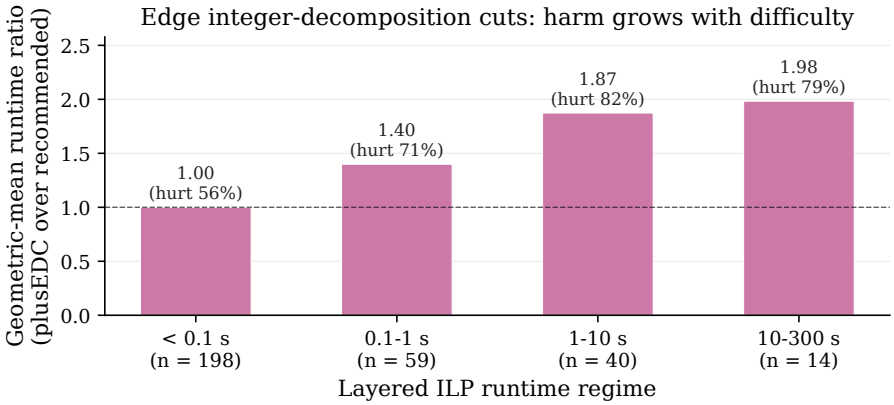


Figure 24: Geometric-mean runtime ratio of the edge integer-decomposition cuts on top of the recommended Layered ILP, broken down by difficulty regime. The harm is negligible on the easiest regime and reaches roughly 2× on the hardest one.

6.8 Ground truth comparison

The previous sections compared the two solvers against each other on the same instances. A separate question is whether the optimal MFDW-SC decomposition recovers the underlying biological transcripts on instances for which the ground truth is known. Of the 325 benchmark instances, only 22 (6.8%) have all distinct ground-truth transcript weights inside the powers-of-two weight set W; on every other instance the biological abundances cannot be expressed as path weights from W, so the MFDW-SC optimum will not match the true transcript set by construction. On the 22 compatible instances both the baseline ILP and the Layered ILP return $k^\star$ equal to the ground-truth number of transcripts and they recover the same multiset of transcript weights. On the remaining 303 instances $k^\star$ for the restricted MFDW-SC problem differs from the number of biological transcripts because the abundances need to be expressed as sums of powers of two. This is a property of the chosen weight set, not of the solver, and it shows that the framework should be re-evaluated on application-specific weight sets before any biological claim is made.

7 Discussion

The starting point of this thesis was a gap between earlier variants of flow decomposition: MFDW restricts the path weights, MFDSC fixes a collection of subpaths that must appear in the decomposition and real RNA splice graphs typically present both features at once. The combined problem MFDW-SC has not been studied before, and a direct combination of the two existing formulations inherits their nonlinear weight-edge products. The path-indexed ILP of Section 3 avoids those products by indexing paths by their weights from the finite set W, yielding a fully linear formulation that retains the identity of individual paths and accommodates both side conditions in one model.

The structural framework of Section 4 reorganizes this formulation around weight equivalence classes and layer graphs. Two consequences drive the rest of the work. The per-edge integer-decomposition number $\pi(e)$ and the weighted antichain bound of Theorem 4.5 give a structural bound that counts, on a single cut of the graph, how many disjoint paths must cross every edge of the cut to carry its flow under W. Independently, the same equivalence classes identify the variables that cannot participate in any feasible solution, which removes them from the model before Gurobi sees it. The starting bound from Theorem 4.5 matches $k^\star$ on the first ILP build for almost every instance, and the variable pruning removes about three quarters of the z variables before the relaxation is solved. Together they give the headline geometric-mean speedup over the baseline ILP. The optional inequalities, by contrast, do not pay for themselves on this benchmark: the variable pruning already implies most of what they would tighten, so adding them grows the constraint matrix without tightening the relaxation.

The same data point to two limitations. The Layered ILP advantage shrinks in the hardest regime, where the instances have more transcripts, fewer feasible weights per edge and less structure to prune; the structural pruning still helps but has less to remove. The ground-truth comparison of Section 6.8 highlights a separate limitation: powers-of-two weights match the biology on only 22 of the 325 instances, so any biological claim about the recovered transcripts requires re-evaluating the framework on application-specific weight sets.

8 Conclusions and future work

Several concrete future directions follow from the limitations seen on the benchmark. The most direct is a head-to-head comparison against the two existing variants, MFDW [4] and MFDSC [16], in the parameter settings where they overlap with MFDW-SC. The MFDW solver matches our framework when the constraint set $\mathcal{R}$ is empty, and the MFDSC solver matches it when the weight set is taken to be $\{1, 2, \ldots, \max_e f(e)\}$. Running all three solvers on a shared benchmark at these aligned settings would isolate the contribution of each side condition and show whether the layered model is competitive in the special cases the dedicated solvers were designed for.

A second direction is to extend the framework to application-specific weight sets. Weight sets derived from observed transcript abundances would close the gap noted in Section 6.8 and may also widen the LP-relaxation margin that the optional inequalities currently fail to exploit. A third direction is to tighten the LP relaxation of the per-edge integer-decomposition constraint directly, so that the EDC inequalities of Section 4.6 become beneficial rather than redundant.

References

[1] Manuel A Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan M 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)*, 2022.

[2] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature Biotechnology*, 34(5):518–524, 2016.

[3] 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.

[4] Andreas Grigorjew, Fernando HC Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I Tomescu. Accelerating ilp solvers for minimum flow decompositions through search space and dimensionality reductions. *arXiv preprint arXiv:2311.10563*, 2024.

[5] Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michael Segalov. How to split a flow? In *IEEE INFOCOM*, pages 828–836, 2012.

[6] Samin Khan, Matti Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I Tomescu. Safety and completeness in flow decompositions for rna assembly. In *RECOMB*, pages 177–192, 2022.

[7] Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Fernando Reidl, Fernando S Villaamil, Blair D Sullivan, and Arjan van der Poel. A practical fpt algorithm for flow decomposition and transcript assembly. In *ALENEX*, pages 75–86, 2018.

[8] Brendan Mumey, Samareh Shahmohammadi, Kaiti McManus, and Sezin Yaman. Parity balancing path flow decomposition and routing. In *Proceedings of the IEEE Globecom Workshops*, pages 1–6, 2015.

[9] Mihaela Pertea, Geo M Pertea, Cornelia 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.

[10] Anthony Rhoads and Kin Fai Au. PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics*, 13(5):278–289, 2015.

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

[12] Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM TCBB*, 16(2):658–670, 2017.

[13] Rory Stark, Marta Grzelak, and James Hadfield. RNA sequencing: the teenage years. *Nature Reviews Genetics*, 20(11):631–656, 2019.

[14] 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.

[15] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, 10(1):57–63, 2009.

[16] Lucia Williams, Alexandru I Tomescu, and Brendan M Mumey. Flow decomposition with subpath constraints. In *WABI*, 2021.

A Complete ILP formulation

A.1 Parameters and indexing

Parameters:

- $k \in \mathbb{Z}_{>0}$: number of path slots (capacity)
- $G = (V, E, f)$: flow network with source s, sink t
- $W = \{w_1, \ldots, w_{|W|}\}$: allowed path weights
- $\mathcal{R} = \{R_1, \ldots, R_m\}$: subpath constraints
- $f : E \to \mathbb{Z}_{>0}$: edge flow function

Index sets:

- $i \in [k] := \{1, \ldots, k\}$: path indices
- $\ell \in [|W|] := \{1, \ldots, |W|\}$: weight indices
- $j \in [m] := \{1, \ldots, m\}$: constraint indices

A.2 Decision variables

$$z_{uvi\ell} \in \{0,1\} \quad \forall (u,v) \in E,\ i \in [k],\ \ell \in [|W|] \quad \text{(edge } (u,v) \text{ used by path } i \text{ with weight } w_\ell)$$
$$y_{i\ell} \in \{0,1\} \quad \forall i \in [k],\ \ell \in [|W|] \quad \text{(path } i \text{ active with weight } w_\ell)$$
$$r_{ij} \in \{0,1\} \quad \forall i \in [k],\ j \in [m] \quad \text{(path } i \text{ satisfies constraint } R_j)$$

A.3 Objective and constraints

Objective: Minimize the number of active path slots

Subject to: the constraints below, with the convention that omitted variables $z_{uvi\ell}$ are fixed to zero. Equivalently, each sum over ℓ may be read as ranging over $\mathcal{L}_{uv} := \{\ell : w_\ell \in W_{\text{valid}}(u, v)\}$, the indices for which Algorithm 8 creates $z_{uvi\ell}$; the unpruned baseline corresponds to $\mathcal{L}_{uv} = [|W|]$ on every edge.

$$\sum_{i=1}^{k} \sum_{\ell=1}^{|W|} w_\ell \cdot z_{uvi\ell} = f(u,v) \qquad \forall (u,v) \in E \qquad \text{(Flow decomp.)}$$

$$\sum_{v:(s,v)\in E} z_{svi\ell} = y_{i\ell} \qquad \forall i \in [k], \ell \in [|W|] \qquad \text{(Source)}$$

$$\sum_{u:(u,t)\in E} z_{uti\ell} = y_{i\ell} \qquad \forall i \in [k], \ell \in [|W|] \qquad \text{(Sink)}$$

$$\sum_{u:(u,v)\in E} z_{uvi\ell} - \sum_{u':(v,u')\in E} z_{vu'i\ell} = 0 \qquad \forall v \in V \setminus \{s,t\}, i \in [k], \ell \in [|W|]$$

(Conservation)

$$\sum_{\ell=1}^{|W|} y_{i\ell} \le 1 \qquad \forall i \in [k] \qquad \text{(Single weight)}$$

$$z_{uvi\ell} \le y_{i\ell} \qquad \forall (u,v) \in E, i \in [k], \ell \in [|W|]$$

(Edge activation)

$$\sum_{u:(u,v)\in E} z_{uvi\ell} \le 1 \qquad \forall v \in V \setminus \{s,t\}, i \in [k], \ell \in [|W|]$$

(In-degree)

$$\sum_{u':(v,u')\in E} z_{vu'i\ell} \le 1 \qquad \forall v \in V \setminus \{s,t\}, i \in [k], \ell \in [|W|]$$

(Out-degree)

$$\sum_{i=1}^{k} r_{ij} \ge 1 \qquad \forall j \in [m] \qquad \text{(Coverage)}$$

$$r_{ij} \le \sum_{\ell=1}^{|W|} y_{i\ell} \qquad \forall i \in [k], j \in [m]$$

(Active path link)

$$\sum_{\ell=1}^{|W|} z_{ei\ell} \ge r_{ij} \qquad \forall i \in [k], j \in [m], e \in R_j$$

(Edge link)

$$\sum_{\ell=1}^{|W|} w_\ell \cdot y_{i\ell} \ge \sum_{\ell=1}^{|W|} w_\ell \cdot y_{(i+1)\ell} \qquad \forall i \in [k-1] \qquad \text{(Sym. breaking)}$$

A.4 Core refinements

The Layered ILP adds three core refinements on top of (16), kept on in every Layered configuration of Section 6 (cf. Section 5.3). The variable $z_{uvi\ell}$ is created only when $w_\ell \in W_{\text{valid}}(u, v)$. For all $i \in [k]$ and $j \in [m]$, the linking inequality

$$r_{ij} \leq \sum_{\ell \in \mathcal{L}_j} y_{i\ell} \quad \text{where } \mathcal{L}_j = \{\ell : w_\ell \in W_{\text{valid}}(R_j)\} \tag{17}$$

restricts each path to weights that are feasible for the constraint it covers. The branching priority of each r_{ij} is set inversely proportional to $|W_{\text{valid}}(R_j)|$ and the iterative search starts from the three-term bound k_0 of Corollary 4.6 computed by Algorithm 16. The five optional inequalities of Section 4.6 are added only when enabled by the corresponding ablation configuration.

A.5 Correctness

Correctness of the formulation (16) is Theorem 3.1 in Section 3: the ILP has a feasible solution with parameter k if and only if there exists a valid MFDW-SC decomposition with at most k paths.

B Algorithms

This appendix presents the algorithms used in the implementation in pseudocode form. Table 7 summarizes the main complexity bounds after the pseudocode.

B.1 Preprocessing

Algorithm 1 Y-to-V reduction

Require: Graph $G = (V, E, f)$, constraints $\mathcal{R}$
Ensure: Reduced graph $G' = (V', E', f')$, mapping $\sigma : E' \to 2^E$

$V_{\text{protected}} \leftarrow \{v \in V : v \text{ is incident to an edge in some } R_j \in \mathcal{R}\}$
$G' \leftarrow G$ and initialize $\sigma(e) \leftarrow [e]$ for every $e \in E$
repeat
    changed $\leftarrow$ false
    for each $v \in V(G') \setminus (\{s, t\} \cup V_{\text{protected}})$ **do**
        if $\deg^-_{G'}(v) = 1$ **then**
            Let (u, v) be the unique incoming edge
            if $(u, w) \notin E(G')$ for every $(v, w) \in E(G')$ **then**
                for each outgoing edge (v, w) **do**
                    Create edge $e' = (u, w)$ with flow $f(v, w)$
                    $\sigma(e') \leftarrow \sigma(u, v)$ followed by $\sigma(v, w)$
                end for
                Delete v and its incident edges from G', insert the shortcut edges
                changed $\leftarrow$ true and **break**
            end if
        else if $\deg^+_{G'}(v) = 1$ **then**
            Let (v, w) be the unique outgoing edge
            if $(u, w) \notin E(G')$ for every $(u, v) \in E(G')$ **then**
                for each incoming edge (u, v) **do**
                    Create edge $e' = (u, w)$ with flow $f(u, v)$
                    $\sigma(e') \leftarrow \sigma(u, v)$ followed by $\sigma(v, w)$
                end for
                Delete v and its incident edges from G', insert the shortcut edges
                changed $\leftarrow$ true and **break**
            end if
        end if
    end for
until changed = false
return G' and σ

Algorithm 2 Identify safe paths

Require: Graph $G = (V, E, f)$
Ensure: Set of safe paths $\mathcal{R}_{\text{safe}}$

$\mathcal{R}_{\text{safe}} \leftarrow \emptyset$
$\mathcal{E}_{\text{visited}} \leftarrow \emptyset$
for each edge $e_1 = (u, v) \in E \setminus \mathcal{E}_{\text{visited}}$ **do**
    $P_{\text{safe}} \leftarrow [e_1]$
    $w_{\text{flow}} \leftarrow f(e_1)$
    $v_{\text{current}} \leftarrow v$
    while $v_{\text{current}} \neq t$ **do**
        $E_{\text{out}} \leftarrow \{(v_{\text{current}}, x) \in E : f(v_{\text{current}}, x) = w_{\text{flow}}\}$
        $E_{\text{in}} \leftarrow \{(x, v_{\text{current}}) \in E : f(x, v_{\text{current}}) = w_{\text{flow}}\}$
        if $|E_{\text{out}}| \neq 1$ **or** $|E_{\text{in}}| \neq 1$ **then**
            break
        end if
        $e_{\text{next}} \leftarrow$ the unique edge in E_{out}
        Append e_{next} to P_{safe}
        $v_{\text{current}} \leftarrow \text{head}(e_{\text{next}})$
    end while
    $u_{\text{current}} \leftarrow u$
    while $u_{\text{current}} \neq s$ **do**
        $E_{\text{in}} \leftarrow \{(x, u_{\text{current}}) \in E : f(x, u_{\text{current}}) = w_{\text{flow}}\}$
        $E_{\text{out}} \leftarrow \{(u_{\text{current}}, x) \in E : f(u_{\text{current}}, x) = w_{\text{flow}}\}$
        if $|E_{\text{in}}| \neq 1$ **or** $|E_{\text{out}}| \neq 1$ **then**
            break
        end if
        $e_{\text{prev}} \leftarrow$ the unique edge in E_{in}
        Prepend e_{prev} to P_{safe}
        $u_{\text{current}} \leftarrow \text{tail}(e_{\text{prev}})$
    end while
    if $|P_{\text{safe}}| \geq 2$ **then**
        $\mathcal{R}_{\text{safe}} \leftarrow \mathcal{R}_{\text{safe}} \cup \{P_{\text{safe}}\}$
        $\mathcal{E}_{\text{visited}} \leftarrow \mathcal{E}_{\text{visited}} \cup P_{\text{safe}}$
    end if
end for
return $\mathcal{R}_{\text{safe}}$

B.2 Structural

Algorithm 3 Compute weight classes

Require: Graph $G = (V, E, f)$, weight set W
Ensure: Representatives $T \cup \{+\infty\}$, classes $\{C_r : r \in T \cup \{+\infty\}\}$, representative function $[\cdot]$

$T \leftarrow \{f(e) : e \in E\}$, sorted in increasing order
for each $w \in W$ **do**
    if $\exists t \in T$ such that $t \geq w$ **then**
        $[w] \leftarrow \min\{t \in T : t \geq w\}$
    else
        $[w] \leftarrow +\infty$
    end if
end for
for each $r \in T \cup \{+\infty\}$ **do**
    $C_r \leftarrow \{w \in W : [w] = r\}$
end for
return T, $\{C_r\}$ and function $[\cdot]$

Algorithm 4 Build layer graphs with reachability

Require: Graph $G = (V, E, f)$, representatives T
Ensure: Layer graphs $\{G_r\}$, reachability sets $\{\mathcal{S}_r, \mathcal{T}_r\}$

for each $r \in T \cup \{+\infty\}$ **do**
    $E_r \leftarrow \emptyset$ if $r = +\infty$, otherwise $\{e \in E : f(e) \geq r\}$
    $G_r \leftarrow (V, E_r)$
    $\mathcal{S}_r \leftarrow \{v \in V : s \rightsquigarrow v \text{ in } G_r\}$ via forward BFS
    $\mathcal{T}_r \leftarrow \{v \in V : v \rightsquigarrow t \text{ in } G_r\}$ via reverse BFS
end for
return $\{G_r\}$ with $\{\mathcal{S}_r\}$ and $\{\mathcal{T}_r\}$

Algorithm 5 Compute feasible weight sets

Require: Representatives T, classes $\{C_r\}$, layers $\{G_r\}$, reachability $\{\mathcal{S}_r, \mathcal{T}_r\}$, constraints $\mathcal{R}$, flows f
Ensure: Feasible sets $\{W_{\text{valid}}(R_j)\}$ and $\{W_{\text{valid}}(u, v)\}$

for each constraint $R_j = (e_1, \ldots, e_p) \in \mathcal{R}$ **do**
  $a_0 \leftarrow$ tail vertex of e_1
  $a_p \leftarrow$ head vertex of e_p
  $U_{R_j} \leftarrow \min\{f(e) : e \in R_j\}$
  $W_{\text{topo}}(R_j) \leftarrow \emptyset$
  for each representative $r \in T \cup \{+\infty\}$ **do**
    if $(R_j \subseteq E_r)$ **and** $(a_0 \in \mathcal{S}_r)$ **and** $(a_p \in \mathcal{T}_r)$ **then**
      $W_{\text{topo}}(R_j) \leftarrow W_{\text{topo}}(R_j) \cup C_r$
    end if
  end for
  $W_{\text{valid}}(R_j) \leftarrow \{w \in W_{\text{topo}}(R_j) : w \le U_{R_j}\}$
end for
for each edge $(u, v) \in E$ **do**
  $W_{\text{valid}}(u, v) \leftarrow \{w \in W : w \le f(u, v),\ u \in \mathcal{S}_{[w]},\ v \in \mathcal{T}_{[w]}\}$
end for
return $\{W_{\text{valid}}(R_j)\}$ and $\{W_{\text{valid}}(u, v)\}$

Algorithm 6 Compute admissible edge set

Require: Layer graph G_r, constraint $R_j = (e_1, \ldots, e_p)$
Ensure: Admissible edge set $\text{Adm}(R_j, r)$

Let $a_0, a_1, \ldots, a_p$ be vertices along R_j where $e_i = (a_{i-1}, a_i)$
$\mathcal{S} \leftarrow \{v \in V : s \rightsquigarrow v \text{ in } G_r\}$ via forward BFS from s
$\mathcal{T} \leftarrow \{v \in V : v \rightsquigarrow t \text{ in } G_r\}$ via backward BFS from t
$\mathcal{P} \leftarrow \{v \in V : v \rightsquigarrow a_0 \text{ in } G_r\}$ via reverse BFS from a_0
$\mathcal{Q} \leftarrow \{v \in V : a_p \rightsquigarrow v \text{ in } G_r\}$ via forward BFS from a_p
$\text{Adm}(R_j, r) \leftarrow R_j$
$\text{Adm}(R_j, r) \leftarrow \text{Adm}(R_j, r) \cup \{(u, v) \in E_r : u \in \mathcal{S},\ v \in \mathcal{P}\}$
$\text{Adm}(R_j, r) \leftarrow \text{Adm}(R_j, r) \cup \{(u, v) \in E_r : u \in \mathcal{Q},\ v \in \mathcal{T}\}$
return $\text{Adm}(R_j, r)$

Algorithm 7 Compute constraint incompatibility graph

Require: Constraints $\mathcal{R}$, per-constraint feasible weight sets $\{W_{\text{valid}}(R_j)\}$, layers $\{G_r\}$
Ensure: Constraint-incompatibility graph $\mathcal{G}_{\text{incomp}}$

$\mathcal{G}_{\text{incomp}} \leftarrow$ empty graph on node set $\mathcal{R}$
for each unordered pair $\{R_i, R_j\} \subseteq \mathcal{R}$ with $i < j$ **do**
    compatible $\leftarrow$ **false**
    for each $w \in W_{\text{valid}}(R_i) \cap W_{\text{valid}}(R_j)$ **do**
        if some s-t path in $G_{[w]}$ contains both R_i and R_j as subpaths **then**
            compatible $\leftarrow$ **true**; **break**
        end if
    end for
    if not compatible **then**
        Add edge $\{R_i, R_j\}$ to $\mathcal{G}_{\text{incomp}}$
    end if
end for
return $\mathcal{G}_{\text{incomp}}$

B.3 ILP construction algorithms

Algorithm 8 Create ILP variables with pruning

Require: Graph G', weights W, classes $\{C_r\}$, reachability $\{\mathcal{S}_r, \mathcal{T}_r\}$, capacity k
Ensure: ILP model with pruned variables

Initialize empty ILP model
$\mathcal{Z} \leftarrow \emptyset$
for $i = 1$ to k **do**
    for each edge $(u, v) \in E'$ **do**
        for $\ell = 1$ to $|W|$ **do**
            $w \leftarrow W[\ell]$
            $r \leftarrow [w]$
            if $w \leq f(u, v)$ **and** $u \in \mathcal{S}_r$ **and** $v \in \mathcal{T}_r$ **then**
                Create binary variable $z_{uvi\ell}$
                $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(u, v, i, \ell)\}$
            end if
        end for
    end for
    for $\ell = 1$ to $|W|$ **do**
        Create binary variable $y_{i\ell}$
    end for
    for $j = 1$ to m **do**
        Create binary variable r_{ij}
    end for
end for
return ILP model with variable set $\mathcal{Z}$

Algorithm 9 Add constraint-weight linking

Require: ILP model, feasible sets $\{W_{\text{valid}}(R_j)\}$, capacity k
Ensure: ILP model with linking constraints
for $i = 1$ to k **do**
  for $j = 1$ to m **do**
    $\mathcal{L}_j \leftarrow \{\ell : W[\ell] \in W_{\text{valid}}(R_j)\}$
    Add constraint: $r_{ij} \leq \sum_{\ell \in \mathcal{L}_j} y_{i\ell}$
  end for
end for
return updated ILP model

Algorithm 10 Add symmetry breaking

Require: ILP model, weight set W, capacity k
Ensure: ILP model with symmetry breaking
for $i = 1$ to $k - 1$ **do**
  Add constraint: $\sum_{\ell=1}^{|W|} W[\ell] \cdot y_{i\ell} \geq \sum_{\ell=1}^{|W|} W[\ell] \cdot y_{(i+1)\ell}$
end for
return updated ILP model

Algorithm 11 Add edge conflict cuts

Require: ILP model, weights W, representative function $[\cdot]$, constraints $\mathcal{R}$, feasible sets $\{W_{\text{valid}}(R_j)\}$, layers $\{G_r\}$, capacity k, threshold τ (default $\tau = 10$)
Ensure: ILP model with conflict cuts
for each constraint $R_j \in \mathcal{R}$ **do**
  if $|W_{\text{valid}}(R_j)| \leq \tau$ **then** ▷ skip constraints with many feasible weights to keep cut count bounded
    for each ℓ such that $W[\ell] \in W_{\text{valid}}(R_j)$ **do**
      $r \leftarrow [W[\ell]]$
      Adm $\leftarrow$ ComputeAdmissibleEdges(G_r, R_j) ▷ Algorithm 6
      for each edge $(u, v) \in E_r \setminus$ Adm **do**
        for $i = 1$ to k **do**
          Add cut: $z_{uvi\ell} \leq 1 - r_{ij}$
        end for
      end for
    end for
  end if
end for
return updated ILP model

Algorithm 12 Add forced single-path constraints

Require: ILP model, constraints $\mathcal{R}$, per-constraint feasible weight sets $\{W_{\text{valid}}(R_j)\}$, layers $\{G_r\}$, capacity k
Ensure: ILP model with forced single-path constraints (Section 4.6)

for each constraint $R_j \in \mathcal{R}$ **do**
  if $|W_{\text{valid}}(R_j)| = 1$ **then**
    $w^\star \leftarrow$ the unique weight in $W_{\text{valid}}(R_j)$, $r \leftarrow [w^\star]$, $\ell^\star \leftarrow$ index of $w^\star$ in W
    $P \leftarrow$ unique s-t path in G_r containing R_j, found by deterministic forward/backward walk
    if P exists **then**
      Choose a fresh slot $i^\star \in [k]$
      Fix $y_{i^\star \ell^\star} \leftarrow 1$
      for each edge $(u, v) \in P$ **do**
        Fix $z_{uvi^\star \ell^\star} \leftarrow 1$
      end for
      Fix $r_{i^\star j} \leftarrow 1$
    end if
  end if
end for
return updated ILP model

Algorithm 13 Add constraint multiplicity upper bounds

Require: ILP model, constraints $\mathcal{R}$, per-constraint feasible weight sets $\{W_{\text{valid}}(R_j)\}$, edge flows f, capacity k
Ensure: ILP model with multiplicity upper bounds (Section 4.6)

for each constraint $R_j \in \mathcal{R}$ **do**
  $w^j_{\min} \leftarrow \min W_{\text{valid}}(R_j)$
  $M_j \leftarrow \min_{e \in R_j} \lfloor f(e)/w^j_{\min} \rfloor$ ▷ multiplicity cap
  if $M_j < k$ **then**
    Add constraint: $\sum_{i=1}^{k} r_{ij} \le M_j$
  end if
end for
return updated ILP model

Algorithm 14 Add edge integer-decomposition cuts

Require: ILP model, per-edge integer-decomposition numbers $\{\pi(u,v)\}$, per-edge valid weight sets $\{W_{\text{valid}}(u,v)\}$, capacity k
Ensure: ILP model with edge integer-decomposition cuts (Section 4.6)

for each edge $(u,v) \in E$ **do**
  $\bar{\pi} \leftarrow \pi(u,v)$
  $b_{uv} \leftarrow \lceil f(u,v) / \max W_{\text{valid}}(u,v) \rceil$
  if $\bar{\pi} > b_{uv}$ **then** ▷ only add when LP relaxation does not already imply the bound
    Add cut: $\sum_{i=1}^{k} \sum_{\ell \,:\, w_\ell \in W_{\text{valid}}(u,v)} z_{uvi\ell} \;\geq\; \bar{\pi}$
  end if
end for
return updated ILP model

Algorithm 15 Add constraint incompatibility cuts

Require: ILP model, constraint-incompatibility graph $\mathcal{G}_{\text{incomp}}$, capacity k
Ensure: ILP model with constraint incompatibility cuts (Section 4.6)

for each edge $\{R_i, R_j\}$ in $\mathcal{G}_{\text{incomp}}$ **do**
  for $i' = 1$ to k **do**
    Add cut: $r_{i'i} + r_{i'j} \leq 1$
  end for
end for
return updated ILP model

B.4 Solving algorithms

Algorithm 16 Compute lower bound

Require: Graph $G = (V, E, f)$, active weights W, per-edge valid weight sets $\{W_{\text{valid}}(u,v)\}$, incompatibility graph $\mathcal{G}_{\text{incomp}}$
Ensure: Lower bound k_0

$F_s \leftarrow \sum_{(s,v)\in E} f(s,v)$
$w_{\max} \leftarrow \max W$
$k_{\text{source}} \leftarrow \lceil F_s / w_{\max} \rceil$
for each $(u,v) \in E$ **do**
  $\pi(u,v) \leftarrow$ minimum number of integer summands to decompose $f(u,v)$ over $W_{\text{valid}}(u,v)$
end for
$\mathcal{A} \leftarrow$ greedy edge antichain in G weighted by $\pi(\cdot)$
$k_{\text{anti}} \leftarrow \sum_{e\in\mathcal{A}} \pi(e)$
$k_{\text{clique}} \leftarrow$ size of the largest incompatibility clique, or 1 if no clique is present
$k_0 \leftarrow \max(k_{\text{source}}, k_{\text{anti}}, k_{\text{clique}})$
return k_0

Algorithm 17 Iterative ILP solver

Require: Preprocessed graph G', structural data, mapping σ
Ensure: Optimal decomposition or not converged

$k_0 \leftarrow \text{ComputeLowerBound}(G', W, \{W_{\text{valid}}(u, v)\}, \mathcal{G}_{\text{incomp}})$ ▷ Algorithm 16
$k_{\text{struct}} \leftarrow |E(G')| - |V(G')| + 2$ ▷ Vatinlen et al. [14] upper bound on decomposition size
$k_{\text{greedy}} \leftarrow$ number of paths in the greedy warm start
$k_{\text{flowUB}} \leftarrow \lceil F_s / \min W \rceil$
$k_{\max} \leftarrow \min\{\text{max\_k\_cap}, \max(k_0, k_{\text{struct}}, k_{\text{greedy}}, k_{\text{flowUB}})\}$
for $k = k_0$ to $k_{\max}$ **do**
    Build refined ILP model for capacity k
    Apply greedy warm start and solve with time limit $T_{\max}$
    if status = OPTIMAL **then**
        Extract active paths and weights from the solution
        if Y-to-V reduction was used **then**
            Expand contracted edges with σ
        end if
        return decomposition
    else if status = INFEASIBLE **then**
        Continue with $k + 1$
    else
        return not converged
    end if
end for
return not converged

Algorithm 18 Extract paths

Require: ILP solution $(z^\star, y^\star, r^\star)$, graph G', capacity k
Ensure: Set of paths $\{P_1, \ldots, P_{k'}\}$ with weights

Paths $\leftarrow \emptyset$
for $i = 1$ to k **do**
    $\ell^\star \leftarrow$ unique ℓ with $y^\star_{i\ell} = 1$ (if exists)
    if $\ell^\star$ exists **then**
        $P_i \leftarrow [s]$
        $v_{\text{current}} \leftarrow s$
        while $v_{\text{current}} \neq t$ **do**
            Find unique $(v_{\text{current}}, v_{\text{next}})$ with $z^\star_{v_{\text{current}}, v_{\text{next}}, i, \ell^\star} = 1$
            Append v_{next} to P_i
            $v_{\text{current}} \leftarrow v_{\text{next}}$
        end while
        Paths $\leftarrow$ Paths $\cup \{(P_i, W[\ell^\star])\}$
    end if
end for
return Paths

B.5 Complete pipeline

Algorithm 19 MFDW-SC solver

Require: Graph $G = (V, E, f)$, weights W, constraints $\mathcal{R}$
Ensure: Optimal flow decomposition

Phase 1: Preprocessing
$(G', \sigma) \leftarrow \text{YtoVReduction}(G, \mathcal{R})$ ▷ Algorithm 1
$\mathcal{R}_{\text{safe}} \leftarrow \text{IdentifySafePaths}(G')$ ▷ Algorithm 2
$\mathcal{R}_{\text{all}} \leftarrow \mathcal{R} \cup \mathcal{R}_{\text{safe}}$

Phase 2: Structural Analysis
$(T, \{C_r\}, [\cdot]) \leftarrow \text{ComputeWeightClasses}(G', W)$ ▷ Algorithm 3
$(\{G_r\}, \{\mathcal{S}_r\}, \{\mathcal{T}_r\}) \leftarrow \text{BuildLayerGraphs}(G', T)$ ▷ Algorithm 4
$(\{W_{\text{valid}}(R_j)\}, \{W_{\text{valid}}(u, v)\}) \leftarrow \text{ComputeFeasibleWeights}(T, \{C_r\}, \{G_r\}, \{\mathcal{S}_r\}, \{\mathcal{T}_r\}, \mathcal{R}_{\text{all}}, f)$ ▷ Algorithm 5
$\mathcal{G}_{\text{incomp}} \leftarrow \text{ComputeIncompatibilityGraph}(\mathcal{R}_{\text{all}}, \{W_{\text{valid}}(R_j)\}, \{G_r\})$ ▷ Algorithm 7

Phases 3 and 4: ILP Construction and Iterative Solving
Solution $\leftarrow \text{IterativeSolver}(G', \text{structural data}, \sigma)$ ▷ Algorithm 17
return Solution

B.6 Complexity summary

Table 7: Algorithm complexity summary

Algorithm	**Complexity**	**Bottleneck**
Y-to-V	$O(\|V\| + \|E\|)$	Graph traversal
Safe paths	$O(\|E\|)$	Edge iteration
Weight classes	$O(\|E\| + \|W\| \log \|T\|)$	Binary search
Layer graphs	$O((\|T\| + 1) \cdot (\|V\| + \|E\|))$	BFS per layer
Feasible sets	$O(m \cdot (\|T\| + 1) \cdot (\|V\| + \|E\|) + \|E\|\|W\|)$	Constraint and edge feasibility
Admissible edges	$O(\|V\| + \|E\|)$	Ordered prefix/suffix reachability
Incompatibility graph	$O(m^2\|W\| \cdot (\|V\| + \|E\|))$	Pairwise compatibility checks
Variable creation	$O(k \cdot \|E\| \cdot \|W\|)$	Triple loop with layer-feasibility pruning
Linking constraints	$O(k \cdot m \cdot \|W\|)$	Constraint addition
Symmetry breaking	$O(k \cdot \|W\|)$	Non-increasing-weight constraint
Edge conflict cuts	$O(k \cdot m \cdot \|W\| \cdot \|E\|)$	Lazy callback in the worst case
Lower bound	$O(\|E\|\|W\| + \|\mathcal{R}\|^2)$	Integer decompositions and clique term
Total preprocessing	$\boldsymbol{O(m \cdot \|T\| \cdot (\|V\| + \|E\|))}$	**Feasibility computation**