

On the Single-Multi-Commodity Gap: Lifting Single- to Multicommodity Flow Instances

Felix P. Broesamle^{1,*} and Stefan Nickel¹

Abstract

Benchmark instances for multicommodity flow problems frequently lack the structural nuances of real-world networks or fail to maintain a rigorous mathematical relationship with their single-commodity counterparts. This paper introduces a formal meta-generation framework that addresses these limitations by lifting single-commodity minimum-cost flow instances into the multicommodity space while strictly preserving the underlying network topology, capacity constraints, and nodal roles. To systematically investigate the effect of commodity-demand heterogeneity (CDH) on computational methods, we propose three distinct integer partitioning schemes: *Uniform*, characterized by homogeneous distributions; *Spread*, introducing high heterogeneity; and *Beta-Binomial*, whose concentration parameter controls the CDH. Central to this approach is the definition of the *(relative) Single-Multi-Commodity Gap*, (r)SMCG, a novel metric that quantifies the divergence in optimal objective values arising from this lifting process, complemented by a formal measure of CDH. Numerical experiments demonstrate that CDH systematically influences the SMCG and affects solver behavior, establishing it as a relevant feature of multicommodity flow instances.

Keywords: Multicommodity Flows, Minimum-Cost Flow, Instance Generation, Linear Programming, Network Optimization, Large-scale Optimization

1 Introduction

A fundamental challenge in network optimization is the transition from single-commodity models to multicommodity systems. While classical single-commodity flows are structurally well-understood and computationally tractable, real-world networks, ranging from telecommunications to global logistics chains, rarely carry a single, uniform type of traffic. Instead, these networks must simultaneously accommodate multiple distinct commodities that share a common infrastructure. This interaction, primarily through the sharing of arc capacities, transforms the problem from a pure network flow into a large-scale linear programming challenge that is notoriously difficult to solve.

The single-commodity Minimum-Cost Flow (MCF) problem is a cornerstone of operations research [1, 2]. Let $G = (V, E)$ be a directed graph, where V is the set of n nodes and $E \subseteq V \times V$ is the set of m directed arcs. Each arc $e = (u, v) \in E$ is characterized by a capacity $u_e \in \mathbb{Q}_+$ and a unit cost $w_e \in \mathbb{R}$. Flow conservation at each node is governed by a nodal demand vector $b \in \mathbb{Z}^n$, satisfying the global balance $\sum_{v \in V} b_v = 0$. By convention, v is a supply node if $b_v > 0$ and a demand node if $b_v < 0$.

For a node $v \in V$, the sets of outgoing and incoming arcs are $\delta^{\text{out}}(v) := \{(v, u) \in E \mid u \in V\}$ and $\delta^{\text{in}}(v) := \{(u, v) \in E \mid u \in V\}$, respectively. The decision variable x_e represents the amount of flow assigned to arc e . The MCF problem is formulated as:

$$\min_{x \in \mathbb{R}^{|E|}} \sum_{e \in E} w_e x_e \tag{1a}$$

$$\text{s.t.} \quad \sum_{e \in \delta^{\text{out}}(v)} x_e - \sum_{e \in \delta^{\text{in}}(v)} x_e = b_v, \quad \forall v \in V, \tag{1b}$$

$$0 \leq x_e \leq u_e, \quad \forall e \in E. \tag{1c}$$

*Corresponding author.

¹Discrete Optimization and Logistics, Institute for Operations Research, Karlsruhe Institute of Technology, Kaiserstraße 89, Karlsruhe, Germany. Email: {felix.broesamle, stefan.nickel}@kit.edu

Date: October 11, 2026.

An instance of the MCF problem is denoted by $\mathcal{P}_S(V, E, u, w, b)$, where $u \in \mathbb{Q}_+^{|E|}$ and $w \in \mathbb{R}^{|E|}$ denote the capacity and cost vectors, respectively.

In the Minimum-Cost Multicommodity Flow (MCMCF) problem, we extend this model by introducing a set of commodities $K = \{1, \dots, K\}$. Each commodity $k \in K$ represents a distinct flow requirement defined by its own demand vector $b^k \in \mathbb{Z}^n$, where $\sum_{v \in V} b_v^k = 0$. We introduce flow variables $x_e^k \in \mathbb{R}_+$ for each commodity k on arc e . Additionally, each commodity flow may incur a specific cost $w_e^k \in \mathbb{R}$ and be subject to an individual capacity $u_e^k \in \mathbb{Q}_+$ in addition to the capacity u_e . The MCMCF problem is formulated as:

$$\min_{x \in \mathbb{R}^{|E| \times K}} \sum_{k=1}^K \sum_{e \in E} w_e^k x_e^k \quad (2a)$$

$$\text{s.t.} \quad \sum_{k=1}^K x_e^k \leq u_e, \quad \forall e \in E, \quad (2b)$$

$$\sum_{e \in \delta^{out}(v)} x_e^k - \sum_{e \in \delta^{in}(v)} x_e^k = b_v^k, \quad \forall v \in V, k \in K, \quad (2c)$$

$$0 \leq x_e^k \leq u_e^k, \quad \forall e \in E, k \in K. \quad (2d)$$

By $\mathcal{P}_M(V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$, we denote an instance of the MCMCF problem.

The formulation provided in (2) is commonly referred to as the *arc-based formulation* of the MCMCF problem [2]. The transition from single-commodity to multicommodity flow introduces a significant increase in structural and computational complexity. Historically, Ford and Fulkerson [3] suggested exploiting the inherent block-angular structure, an insight later formalized by Dantzig and Wolfe through their decomposition principle [4]. These methodologies underpin modern *branch-and-cut-and-price* frameworks [5] and solvers such as GCG [6] and Coluna [7]. Comprehensive surveys on algorithms for the MCMCF are provided by [8,9]. For specialized solution strategies, including interior point methods and decomposition methods, we refer to [10–12].

Recent advances include GPU-based interior point methods [13] and a refined polynomial reduction of general linear programs to maximum 2-commodity flow problems [14,15].

Related Work: Parallel to algorithmic advances, the development of standardized test instances and file formats has been crucial. A comprehensive overview of benchmarks for both MCF and MCMCF is provided by Frangioni [16,17]. For single-commodity flows, the well-known NETGEN [18,19] and GridGen (developed by Lee and Orlin [19,20]) generators, among others, have served as primary sources for decades, alongside specialized instances and established data formats, such as the DIMACS [21] .min format.

For MCMCF, researchers have traditionally relied on real-world problems, such as the Patient Distribution System (PDS) instances, derived from military medical evacuation scenarios and described in [22], or the widely utilized MNETGEN generator [23] and its refined variant [17,24]. A meta-generation approach based on stochastic scaling has been investigated in [25] and further improved in the dimacs2pprn meta-generator by Frangioni [17,24]. While these generators are valuable, they typically do not allow direct control over the relationship between a given single-commodity instance and the resulting multicommodity instances, nor do they provide a systematic way to vary commodity-demand heterogeneity (CDH), which we introduce formally in Section 2. The proposed framework is specifically designed to fill this gap by preserving the underlying network and its single-commodity structure, and by providing control over CDH.

The importance of multicommodity flow instances as benchmarks for general-purpose linear programming (LP) solvers is well documented, e.g., by Bixby [26].

Our Contribution: This work introduces a *meta-generation framework* designed to strictly maintain the network topology and nodal roles of an underlying MCF instance by partitioning the nodal supply and demand across K commodities, while providing direct control over CDH. The contribution is fourfold:

1. The meta-generation framework is built upon integer partitioning. Therefore, we propose three distinct integer partitioning strategies: *Uniform*, characterized by homogeneous commodity distributions, *Spread*, introducing high heterogeneity, and *Beta-Binomial*, whose concentration parameter provides control over CDH.
2. We define the *(relative) Single-Multi-Commodity Gap*, (r)SMCG, as a formal metric to quantify the difference in the optimal objective value arising from the transformation, and establish theoretical results concerning the feasibility and optimal values. We further introduce a formal measure of CDH, which allows a comparison of the partitioning strategies.
3. We provide a comprehensive computational study across multiple solution methods and different base network generators.
4. To facilitate further research and reproducibility, the framework is implemented in **Rust** and released as the open-source **Python** library **s2mflow** [27].

The computational experiments in Section 4 reveal that solution methods exhibit different scaling behaviors depending on the chosen partitioning strategy. These findings demonstrate that CDH is not merely a parameter, but a critical structural characteristic for the systematic classification of MCMCF benchmark instances and the evaluation of solution methods.

The remainder of this work is organized as follows: In Section 2, we introduce the meta-generation framework. Section 3 provides the theoretical analysis of the resulting problem properties and the SMCG. Section 4 presents numerical experiments demonstrating the impact of CDH on solver performance and the SMCG.

2 The Meta-Generation Framework

The primary challenge in deriving a multicommodity instance $\mathcal{P}_M$ from a single-commodity instance $\mathcal{P}_S$ lies in the consistent partitioning of the demand vector $b \in \mathbb{Z}^n$. We introduce a meta-generation framework that transforms the nodal demands into a commodity-demand matrix $B \in \mathbb{Z}^{n \times K}$, where the entry B_{vk} denotes the demand of commodity k at node $v \in V$. Within this framework, the matrix B and the set of column vectors $\{b^k\}_{k \in K}$ are used interchangeably, such that each $b^k = B_{:,k} \in \mathbb{Z}^n$ defines the individual demand vector for commodity $k \in K$.

To ensure that the resulting instance $\mathcal{P}_M$ inherits the structural characteristics of $\mathcal{P}_S$, we impose the following requirements:

$$\text{sign}(B_{vk}) \in \{0, \text{sign}(b_v)\}, \quad \forall v \in V, \forall k \in K, \quad (3a)$$

$$B_{vk} \in \mathbb{Z}, \quad \forall v \in V, \forall k \in K, \quad (3b)$$

$$\sum_{k=1}^K B_{vk} = b_v, \quad \forall v \in V, \quad (3c)$$

$$\sum_{v \in V} B_{vk} = 0, \quad \forall k \in K. \quad (3d)$$

Constraint (3a) ensures role preservation: a supply node ($b_v > 0$) never becomes a demand node; conversely, a demand node ($b_v < 0$) never becomes a supply node. Constraint (3c) enforces aggregate consistency, while (3d) ensures global flow conservation for each commodity. By maintaining integrality (3b), we guarantee that an instance $\mathcal{P}_S$ with integral demand data yields a multicommodity instance $\mathcal{P}_M$ with integral demand data. This preservation is computationally significant. When applying Dantzig-Wolfe decomposition or column generation, it ensures that the resulting subproblems remain single-commodity minimum-cost flow problems with integral supply and demand.

It follows directly from (3a) that for any transshipment node $v \in V$ with $b_v = 0$, the commodity-specific demands must satisfy $B_{vk} = 0$ for all $k \in K$. Thus, the structural role of a transshipment node is invariant under the proposed transformation.

For the remainder of this section, we restrict our algorithmic focus to the set of supply and demand nodes $\tilde{V} = \{v \in V : b_v \neq 0\}$. For notational brevity, we let $b \in \mathbb{Z}^{\tilde{V}}$ denote the restricted demand vector and $B \in \mathbb{Z}^{\tilde{V} \times K}$ the resulting multicommodity demand matrix. Nodes $v \notin \tilde{V}$ are implicitly assigned $B_{vk} = 0$ for all $k \in K$.

The framework proceeds in two phases:

1. *Local Partitioning*: An initial distribution of nodal demands to satisfy (3a), (3b) and (3c).
2. *Global Balancing*: A redistribution phase to restore global flow conservation and satisfy (3d).

The framework operates on integral demands to ensure exact flow conservation without numerical round-off errors. Extending the methodology to non-integral data can in principle be achieved by sampling from continuous probability distributions and using a numerically careful adaptation of the global balancing phase.

2.1 Phase 1: Local Partitioning Strategies

We propose three distinct integer partitioning strategies to control the degree of CDH, which is formally defined in Section 2.2. The *Uniform* and *Spread* strategies represent the two extremes of CDH, while the *Beta-Binomial* strategy provides a controllable level of it.

2.1.1 Uniform Integer Partitioning

The *Uniform* strategy minimizes heterogeneity by distributing b_v as evenly as possible among the K commodities.

Algorithm 1 Integer Partitioning: *Uniform*

Input: Vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, and $K \in \mathbb{N}$.

Output: Matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c).

- 1: Initialize $B \in \mathbb{Z}^{n \times K}$ with $B_{ik} \leftarrow \text{sign}(b_i) \lfloor \frac{|b_i|}{K} \rfloor$ for all $i = 1, \dots, n, k = 1, \dots, K$.
 - 2: Let $next = 1$ ▷ Global remainder pointer.
 - 3: **for** $i = 1 : n$ **do**
 - 4: $r \leftarrow |b_i| \bmod K$.
 - 5: **for** $v = 1 : r$ **do**
 - 6: Update $B_{i,next} \leftarrow B_{i,next} + \text{sign}(b_i)$.
 - 7: Update $next \leftarrow (next \bmod K) + 1$.
 - 8: **return** B .
-

Theorem 2.1. Given a vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, and $K \in \mathbb{N}$, Algorithm 1 terminates in $\mathcal{O}(nK)$ time and produces a matrix $B \in \mathbb{Z}^{n \times K}$ such that for each node i , $\sum_{k=1}^K B_{ik} = b_i$, $\text{sign}(B_{ik}) \in \{0, \text{sign}(b_i)\}$, and $B_{ik} \in \mathbb{Z}$ for all $k \in \{1, \dots, K\}$.

Proof. Algorithm 1 iterates through the n entries of b . For each $i \in \{1, \dots, n\}$, the initialization of row B_i requires K operations. Since the remainder r satisfies $0 \leq r < K$, the subsequent distribution loop performs at most $K - 1$ incremental updates. Consequently, the total computational effort is proportional to the number of entries of matrix B , yielding a complexity of $\mathcal{O}(nK)$.

Sum Preservation: For any $i = 1, \dots, n$, let $|b_i| = q_i K + r_i$, where $q_i = \lfloor \frac{|b_i|}{K} \rfloor$ and $0 \leq r_i < K$. The algorithm initializes the row B_i such that each of the K entries is $\text{sign}(b_i)q_i$. It then selects r_i entries to be updated by adding $\text{sign}(b_i)$. The resulting row sum is:

$$\begin{aligned}
 \sum_{k=1}^K B_{ik} &= r_i (\text{sign}(b_i)q_i + \text{sign}(b_i)) + (K - r_i) (\text{sign}(b_i)q_i) \\
 &= \text{sign}(b_i) (r_i(q_i + 1) + Kq_i - r_iq_i) = \text{sign}(b_i) (r_iq_i + r_i + Kq_i - r_iq_i) \\
 &= \text{sign}(b_i) (Kq_i + r_i) = \text{sign}(b_i)|b_i| = b_i.
 \end{aligned}$$

Sign Consistency: Since $q_i \geq 0$, the base assignment $\text{sign}(b_i)q_i$ and the optional increment $\text{sign}(b_i)$ always maintain the sign of b_i or result in zero, satisfying (3a). *Integrality:* Since B_{ik} is initialized via the floor function $\lfloor \cdot \rfloor$ and subsequently modified only by unit increments, $B_{ik} \in \mathbb{Z}$ for all i, k . $\square$

Example 2.1. Consider the vector $b = [7, 10, 4, -7, -10, -4]$, which satisfies $\sum_i b_i = 0$, with $K = 3$. Applying Algorithm 1, we first perform the base assignment $B_{ik} = \text{sign}(b_i) \lfloor \frac{|b_i|}{K} \rfloor$, yielding:

$$B = \begin{pmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \\ 1 & 1 & 1 \\ -2 & -2 & -2 \\ -3 & -3 & -3 \\ -1 & -1 & -1 \end{pmatrix}.$$

The algorithm then processes the remainder $r = [1, 1, 1, 1, 1, 1]$. Since each remainder equals 1, the pointer cyclically updates the columns:

$$B = \begin{pmatrix} 3 & 2 & 2 \\ 3 & 4 & 3 \\ 1 & 1 & 2 \\ -3 & -2 & -2 \\ -3 & -4 & -3 \\ -1 & -1 & -2 \end{pmatrix}.$$

Summing the columns, we observe $\sum_i B_{i,k} = 0$ for all $k \in \{1, 2, 3\}$. In this specific case, (3) is satisfied and no further work is necessary.

2.1.2 Spread Integer Partitioning

In contrast to the previous method *Uniform*, the *Spread* method is designed to introduce large commodity supply and demand heterogeneity.

Algorithm 2 Integer Partitioning: *Spread*

Input: Vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, and $K \in \mathbb{N}$.

Output: Matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c).

- 1: Initialize $B = 0_{n \times K}$.
 - 2: **for** $i = 1 : n$ **do**
 - 3: Set $a \leftarrow |b_i|$.
 - 4: Sample $K - 1$ integers $C = \{c_1, \dots, c_{K-1}\}$ uniformly and independently at random from the set $\{0, \dots, a\}$ with replacement.
 - 5: Sort $\{0, a\} \cup C$ such that $0 = c_0 \leq c_1 \leq \dots \leq c_{K-1} \leq c_K = a$.
 - 6: **for** $k = 1 : K$ **do**
 - 7: Update $B_{ik} \leftarrow \text{sgn}(b_i) \cdot (c_k - c_{k-1})$.
 - 8: **return** B .
-

Theorem 2.2. Given a vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, and $K \in \mathbb{N}$, Algorithm 2 produces a matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c). Assuming that sampling $K - 1$ values uniformly and independently at random from $\{0, \dots, a\}$ with replacement where $a \in \mathbb{N}$ requires $\mathcal{O}(K)$ time, the algorithm terminates in $\mathcal{O}(nK \log(K))$ time.

Proof. Algorithm 2 iterates through n entries. For each entry, sampling $K - 1$ values takes $\mathcal{O}(K)$ by assumption and sorting the $K + 1$ values takes $\mathcal{O}(K \log(K))$. The subsequent row assignment requires $\mathcal{O}(K)$ operations. Therefore, the computational effort is $\mathcal{O}(nK \log(K))$.

Sum Preservation: For any row i , the row entries are defined as the differences of a sorted sequence $c_0, \dots, c_K$. By a telescoping sum argument:

$$\sum_{k=1}^K B_{ik} = \sum_{k=1}^K \text{sign}(b_i)(c_k - c_{k-1}) = \text{sign}(b_i) \sum_{k=1}^K (c_k - c_{k-1}) = \text{sign}(b_i)(c_K - c_0).$$

Since $c_k = |b_i|$ and $c_0 = 0$ by construction, we have $\sum_{k=1}^K B_{ik} = \text{sign}(b_i)|b_i| = b_i$, satisfying (3c).

Sign Consistency: For each $k \in \{1, \dots, K\}$, the value $(c_k - c_{k-1})$ is non-negative because the sequence $(c_j)_{j \in \{0, \dots, K\}}$ is sorted. Consequently, $B_{ik} \geq 0$ if $b_i > 0$ and $B_{ik} \leq 0$ if $b_i < 0$, satisfying (3a).

Integrity: The matrix entries are defined as differences of integers, thus, B_{ik} is integer-valued. $\square$

The generation strategy provided in Algorithm 2 induces high commodity heterogeneity and is readily adaptable to alternative probability distributions.

Example 2.2. Consider $b = (13, 2, 5, -6, -14)$ and $K = 3$. We apply Algorithm 2:

- $i = 1$ ($b_1 = 13$): Sample $K - 1 = 2$ integers from $\{0, \dots, 13\}$, e.g., $C = \{3, 11\}$. The sorted sequence is $\{0, 3, 11, 13\}$, yielding $B_{1,\cdot} = (3 - 0, 11 - 3, 13 - 11) = (3, 8, 2)$.
- $i = 2$ ($b_2 = 2$): Sample 2 integers from $\{0, \dots, 2\}$, e.g. $C = \{0, 1\}$. The sorted sequence is $\{0, 0, 1, 2\}$, yielding $B_{2,\cdot} = (0, 1, 1)$.
- $i = 3, 4, 5$: Following the same logic, we assume we obtain the following matrix:

$$B = \begin{pmatrix} 3 & 8 & 2 \\ 0 & 1 & 1 \\ 2 & 0 & 3 \\ -1 & -4 & -1 \\ -2 & -9 & -3 \end{pmatrix}.$$

Calculating the column sums reveals a global imbalance: columns 1 and 3 exhibit a surplus, while column 2 is in a deficit. This state of global imbalance necessitates the subsequent balancing phase described in Algorithm 4.

2.1.3 Beta-Binomial Integer Partitioning

The *Beta-Binomial* strategy produces instances with an adjustable level of CDH through a concentration parameter $c > 0$.

Algorithm 3 Integer Partitioning: *Beta-Binomial*

Input: Vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, $K \in \mathbb{N}$, and the concentration parameter $c > 0$.

Output: Matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c).

- 1: Initialize $B \in \mathbb{Z}^{n \times K}$ with $B_{ik} \leftarrow 0$ for all $i = 1, \dots, n$, $k = 1, \dots, K$.
 - 2: **for** $i = 1 : n$ **do**
 - 3: Set $a \leftarrow |b_i|$ and $r \leftarrow a$.
 - 4: **for** $j = 1 : K - 1$ **do**
 - 5: Sample $X_j \sim \text{BetaBinomial}(r, c, c \cdot (K - j))$.
 - 6: Update $B_{i,j} \leftarrow \text{sign}(b_i) \cdot X_j$.
 - 7: Update $r \leftarrow r - X_j$.
 - 8: Update $B_{i,K} \leftarrow \text{sign}(b_i) \cdot r$.
 - 9: **return** B .
-

Theorem 2.3. Given a vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \dots, n$, $K \in \mathbb{N}$, and $c > 0$, Algorithm 3 produces a matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c). Assuming that sampling from a beta-binomial distribution requires expected $\mathcal{O}(1)$ time, the algorithm terminates in expected $\mathcal{O}(nK)$ time.

Proof. Algorithm 3 iterates through the n entries of b . For each entry, it performs $K - 1$ beta-binomial samples. By assumption, each sample takes expected constant time. Hence, the total runtime is $\mathcal{O}(nK)$.

Sum Preservation: For any node i , after the inner loop, the remaining absolute demand is

$$r = a - \sum_{j=1}^{K-1} X_j.$$

Therefore,

$$\sum_{k=1}^K B_{ik} = \text{sign}(b_i) \left(\sum_{j=1}^{K-1} X_j + r \right) = \text{sign}(b_i) a = b_i,$$

which satisfies (3c).

Sign Consistency: Every X_j and the final remaining value are non-negative integers. Hence, if $b_i > 0$, then $B_{ik} \geq 0$ for all k , and if $b_i < 0$, then $B_{ik} \leq 0$ for all k . Thus, (3a) holds.

Integrality: Since beta-binomial samples and the remaining values are integers, $B_{ik} \in \mathbb{Z}$, satisfying (3b). $\square$

Remark 1. The parameter choice $\alpha = c$ and $\beta_j = c(K - j)$, $c > 0$ ensures that, in expectation, each commodity receives the same share a/K of a nodal demand. For large c , the *Beta-Binomial* method behaves close to the *Uniform* method; for small c , e.g. $c = 1$, it behaves close to the *Spread* method. The computational experiments in Section 4.2.1 and Figure 1 illustrate this in detail. Thus the *Beta-Binomial* method provides a bridge between *Uniform* and *Spread*, with c serving as a control parameter for CDH.

2.2 Commodity-Demand Heterogeneity

To compare the three partitioning strategies, we introduce a scalar measure of heterogeneity. As transshipment nodes are irrelevant for CDH, they are excluded from the definition.

Definition 2.4 (Commodity-Demand Heterogeneity). For a demand vector $b \in \mathbb{Z}^{|\tilde{V}|}$ with $b_v \neq 0$ for all $v \in \tilde{V}$, and a corresponding commodity-demand matrix $B \in \mathbb{Z}^{|\tilde{V}| \times K}$ satisfying (3), the commodity-demand heterogeneity (CDH) is defined as

$$\text{CDH}(B) = \frac{1}{|\tilde{V}|} \sum_{v \in \tilde{V}} \frac{K}{2(K-1)} \sum_{k=1}^K \left| \frac{|B_{vk}|}{|b_v|} - \frac{1}{K} \right|. \quad (4)$$

For each active node v , the values $|B_{vk}|/|b_v|$ form a probability distribution over the commodities. The inner term,

$$\frac{1}{2} \sum_{k=1}^K \left| \frac{|B_{vk}|}{|b_v|} - \frac{1}{K} \right|,$$

is the total variation distance between that distribution and the uniform distribution $(1/K, \dots, 1/K)$. The factor $K/(K-1)$ normalizes this distance to the unit interval. Hence $\text{CDH}(B)$ is the average normalized deviation from a perfectly even split.

Lemma 2.5. For every matrix B satisfying (3), it holds

$$0 \leq \text{CDH}(B) \leq 1.$$

Proof. We fix an active node $v \in \tilde{V}$, and set

$$p_{vk} = \frac{|B_{vk}|}{|b_v|}, \quad k = 1, \dots, K.$$

We have $p_{vk} \geq 0$, and by (3a) and (3c), $\sum_{k=1}^K p_{vk} = 1$. Thus $p_v = (p_{v1}, \dots, p_{vK})$ is a probability distribution.

The total variation distance between p_v and the uniform distribution $u = (1/K, \dots, 1/K)$ is

$$d_{\text{TV}}(p_v, u) = \frac{1}{2} \sum_{k=1}^K \left| p_{vk} - \frac{1}{K} \right|.$$

It is well known that $0 \leq d_{\text{TV}}(p_v, u) \leq 1 - 1/K$. The upper bound is attained when p_v is a point mass on one commodity, in which case

$$d_{\text{TV}}(p_v, u) = \frac{1}{2} \left[\left(1 - \frac{1}{K}\right) + (K-1) \cdot \frac{1}{K} \right] = 1 - \frac{1}{K}.$$

Therefore,

$$0 \leq \frac{K}{2(K-1)} \sum_{k=1}^K \left| p_{vk} - \frac{1}{K} \right| \leq 1.$$

Averaging over all $v \in \tilde{V}$ yields $0 \leq \text{CDH}(B) \leq 1$. $\square$

Remark 2. The three integer partitioning methods introduced in Section 2.1 differ in $\text{CDH}(B)$. The *Uniform* method yields values close to zero, while the *Spread* method generally yields high values, see Section 4. The *Beta-Binomial* method provides control over $\text{CDH}(B)$ through the concentration parameter c , enabling the generation of instances with an intermediate level. Thus $\text{CDH}(B)$ provides a basis for comparing the partitioning methods.

2.3 Phase 2: Balancing

While Phase 1 ensures that nodal demands are locally consistent, the global balance requirement (3d) is likely violated, resulting in a non-zero commodity imbalance $\Delta_k := \sum_{i=1}^n B_{i,k}$. To restore balance, i.e., $\Delta_k = 0$ for all $k \in K$, we introduce the following balancing algorithm.

Algorithm 4 reallocates demand units between a surplus commodity sk and a deficit commodity dk at a specific node (row) i . We remark that as long as there is a commodity with a surplus, there must be a commodity with a deficit. A swap is feasible if it satisfies the role-preservation constraint (3a). By utilizing persistent pointers for both rows (nodes) and columns (commodities) (p_{node} and p_{def}), the algorithm ensures that the adjustments are distributed uniformly across the network, preventing the over-concentration of changes in a small subset of nodes.

Theorem 2.6. Given an initial matrix $B \in \mathbb{Z}^{n \times K}$ and a vector $b \in \mathbb{Z}^n$, where $\sum_{i=1}^n b_i = 0$, $b_i \neq 0$, such that B satisfies (3a)–(3c), Algorithm 4 terminates and returns a matrix that satisfies (3).

Proof. We first show that Algorithm 4 preserves (3a)–(3c), followed by proving the termination and correctness in achieving (3).

Row-Sum Preservation: Each update performs a simultaneous increment, $B_{i,sk} \leftarrow B_{i,sk} - 1$, and decrement, $B_{i,dk} \leftarrow B_{i,dk} + 1$, on the same row i (p_{node}), resulting in a row-sum change of zero.

Sign Consistency: Since $b_i \neq 0$, we analyze two cases. For $b_i > 0$, $B_{i,\cdot}$ entries are non-negative. $B_{i,sk}$ is only decremented if $B_{i,sk} > 0$, and $B_{i,dk}$ is only incremented, thus staying non-negative. For $b_i < 0$, $B_{i,\cdot}$ entries are non-positive. The entry $B_{i,dk}$ is only incremented if $B_{i,dk} < 0$, and $B_{i,sk}$ is only decremented, preserving non-positivity.

Integrality: The algorithm preserves integrality as all reallocations are performed via unit swaps between surplus and deficit commodities.

Termination: We define the global imbalance potential as the total surplus in the system: $\Phi(B) = \sum_{k=1}^K \max\{0, \Delta_k\} = \frac{1}{2} \sum_{k=1}^K |\Delta_k|$, where $\Delta_k = \sum_{i=1}^n B_{i,k}$ for each $k \in K$. Since $\sum_{i=1}^n b_i = 0$ and the row-sum property $\sum_{k=1}^K B_{i,k} = b_i$ is preserved, the total system imbalance is zero, i.e., $\sum_{k=1}^K \Delta_k = 0$. This implies that a surplus in one commodity ($\Delta_{sk} > 0$) necessitates a corresponding deficit in another commodity ($\Delta_{dk} < 0$).

Algorithm 4 Global Balancing

Input: Matrix $B \in \mathbb{Z}^{n \times K}$, nodal demands $b \in \mathbb{Z}^n$, number of commodities $K \in \mathbb{N}$.

Output: Matrix B satisfying (3).

```
1: Initialize  $\Delta_k \leftarrow \sum_{i=1}^n B_{ik}$  for all  $k \in K$ .
2: Initialize  $p_{node} \leftarrow 1$  and  $p_{def} \leftarrow 1$ . ▷ Node and deficit commodity pointers.
3: while  $\exists k : \Delta_k > 0$  do
4:   Let  $\mathcal{K}^+ \leftarrow (k_1, \dots, k_m)$  be a sequence of indices  $\{k \mid \Delta_k > 0\}$ .
5:   Let  $\mathcal{K}^- \leftarrow (k_1, \dots, k_q)$  be a sequence of indices  $\{k \mid \Delta_k < 0\}$ .
6:   if  $p_{def} > |\{k \mid \Delta_k < 0\}|$  then
7:     Set  $p_{def} = 1$ .
8:   Sample  $sk$  uniformly at random from  $\mathcal{K}^+$ . ▷ Pick surplus commodity to balance.
9:   for  $step = 1 : n$  do
10:    Set  $i \leftarrow p_{node}$ .
11:    for  $attempt = 1 : |\mathcal{K}^-|$  do
12:      Set  $dk \leftarrow \mathcal{K}_{p_{def}}^-$ .
13:      Set  $can\_swap \leftarrow (b_i > 0 \wedge B_{i,sk} > 0) \vee (b_i < 0 \wedge B_{i,dk} < 0)$ .
14:      Update  $p_{def} \leftarrow (p_{def} \bmod |\mathcal{K}^-|) + 1$ .
15:      if  $can\_swap$  then
16:        Update  $B_{i,sk} \leftarrow B_{i,sk} - 1$ , and  $B_{i,dk} \leftarrow B_{i,dk} + 1$ .
17:        Update  $\Delta_{sk} \leftarrow \Delta_{sk} - 1$ , and  $\Delta_{dk} \leftarrow \Delta_{dk} + 1$ .
18:        break
19:      end for
20:    Update  $p_{node} \leftarrow (p_{node} \bmod n) + 1$ .
21:    if  $\Delta_{sk} = 0$  then
22:      break
23:    end for
24: end while
25: return  $B$ 
```

Each successful swap reduces the surplus Δ_{sk} by exactly one unit and increases the deficit Δ_{dk} by one unit (decreasing $|\Delta_{dk}|$). Consequently, the potential $\Phi(B)$ strictly decreases by 1 per swap.

As long as $\Phi(B) > 0$, the existence of a valid swap node i is guaranteed by the fact that b is balanced ($\sum_{i=1}^n b_i = 0$) and the role-preservation constraint, which ensures that units can always be redistributed without violating nodal sign. Upon termination, $\Phi(B) = 0$ holds, and global balance is satisfied. $\square$

We now continue Example 2.2 from the previous section to illustrate the balancing phase.

Example 2.3. We continue from Example 2.2, where the *Spread* partitioning resulted in the initial matrix B with imbalance $\Delta = (2, -4, 2)$ for the vector $b = (13, 2, 5, -6, -14)$:

$$B = \begin{pmatrix} 3 & 8 & 2 \\ 0 & 1 & 1 \\ 2 & 0 & 3 \\ -1 & -4 & -1 \\ -2 & -9 & -3 \end{pmatrix}.$$

We initialize $p_{node} = 1$ and $p_{def} = 1$. The surplus and deficit sets are $\mathcal{K}^+ = (1, 3)$ and $\mathcal{K}^- = (2)$.

- Iteration 1: Assume the random choice yields $sk = 1$. For $p_{node} = 1$, node 1 is a supply node ($b_1 > 0$) and $B_{1,1} = 3 > 0$. The swap with $dk = 2$ is feasible. Update: $B_{1,1} \leftarrow 2, B_{1,2} \leftarrow 9, \Delta \leftarrow (1, -3, 2)$. Pointers: $p_{def} = 1, p_{node} = 2$.
- Iteration 2: For $sk = 1, p_{node} = 2$, node 2 has $B_{2,1} = 0$; no swap possible. Update $p_{def} = 1, p_{node} = 3$.

- Iteration 3: For $sk = 1, p_{node} = 3$, node 3 has $B_{1,3} = 2 > 0$, swap is feasible. Update: $B_{3,1} \leftarrow 1, B_{3,2} \leftarrow 1, \Delta \leftarrow (0, -2, 2)$. As $\Delta_1 = 0$, the loop for $sk = 1$ terminates. Update $p_{node} = 4$.
- Iteration 4: Now $\mathcal{K}^+ = (3)$, so $sk = 3$. For $p_{node} = 4$, node 4 is a demand node ($b_4 < 0$) and $B_{4,2} = -4 < 0$, swap feasible. Update $B_{4,3} \leftarrow -2, B_{4,2} \leftarrow -3, \Delta \leftarrow (0, -1, 1)$. Update $p_{node} = 5$.
- Iteration 5: For $sk = 3, p_{node} = 5$, node 5 has $B_{5,2} = -9 < 0$, swap feasible. Update: $B_{5,3} \leftarrow -4, B_{5,2} \leftarrow -8, \Delta \leftarrow (0, 0, 0)$.

The algorithm terminates as $\Phi(B) = 0$. The final balanced matrix is:

$$B = \begin{pmatrix} 2 & 9 & 2 \\ 0 & 1 & 1 \\ 1 & 1 & 3 \\ -1 & -3 & -2 \\ -2 & -8 & -4 \end{pmatrix}$$

By inspection, $\sum_{k \in K} B_{ik} = b_i$ for all i and $\sum_{i=1}^n B_{ik} = 0$ for all k , satisfying all requirements of (3).

Remark 3. Let MCF instance $\mathcal{P}_S(V, E, u, w, b)$ be given. For $K \geq 2$, let the corresponding MCMCF instance be given by $\mathcal{P}_M(V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$, where $u^k = u$ for all $k \in K$ and with a corresponding commodity demand matrix B satisfying (3). Note that the MCMCF instance is not necessarily feasible.

3 Theoretical Properties

In this section, we establish the fundamental theoretical properties linking the generated multicommodity instance $\mathcal{P}_M$ to its single-commodity origin $\mathcal{P}_S$.

Proposition 3.1. Given a single-commodity instance $\mathcal{P}_S = (V, E, u, w, b)$, let the corresponding multicommodity instance be $\mathcal{P}_M = (V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$, where $\{b^k\}_{k \in K}$ satisfies (3). If $\mathcal{P}_M$ is feasible, then $\mathcal{P}_S$ is feasible.

Proof. Let $\{f^k\}_{k \in K}$ be a feasible multicommodity flow for $\mathcal{P}_M$, where $f^k \in \mathbb{R}_{\geq 0}^{|E|}$ denotes the flow vector for commodity $k \in K$. We define the aggregated flow vector $\tilde{x} \in \mathbb{R}_{\geq 0}^{|E|}$ such that $\tilde{x}_e = \sum_{k \in K} f_e^k$ for all $e \in E$.

First, $\tilde{x}$ satisfies the single-commodity capacity constraints (1c) directly from the bundle capacity constraint (2b) of $\mathcal{P}_M$:

$$0 \leq \tilde{x}_e = \sum_{k \in K} f_e^k \leq u_e, \quad \forall e \in E.$$

Second, $\tilde{x}$ satisfies the single-commodity flow conservation (1b). Using the multicommodity flow conservation (2c) and property (3c), we obtain for every node $v \in V$:

$$\begin{aligned} \sum_{e \in \delta^{out}(v)} \tilde{x}_e - \sum_{e \in \delta^{in}(v)} \tilde{x}_e &= \sum_{k \in K} \left(\sum_{e \in \delta^{out}(v)} f_e^k - \sum_{e \in \delta^{in}(v)} f_e^k \right) \\ &\stackrel{(2c)}{=} \sum_{k \in K} b_v^k \stackrel{(3c)}{=} b_v. \end{aligned}$$

Therefore, the single-commodity instance is feasible. $\square$

The individual commodity capacities $\{u^k\}_{k \in K}$ do not affect the validity of Proposition 3.1. However, to isolate the impact of introducing multiple commodities without conflating this structural change with additional resource restrictions, we focus on instances where individual capacities do not further restrict the feasible region.

Definition 3.2 ((Relative) Single-Multi-Commodity Gap). Let $\mathcal{P}_S = (V, E, u, w, b)$ be a feasible MCF instance with optimal objective value z^S , and let $\mathcal{P}_M = (V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$ be the corresponding MCMCF instance, where $u^k = u$ and $w^k = w$ for all $k \in K$, and $\{b^k\}_{k \in K}$ satisfies (3). Let z^M be the optimal objective value of $\mathcal{P}_M$. We define the Single-Multi-Commodity Gap SMCG as

$$\text{SMCG} := z^M - z^S,$$

and the relative Single-Multi-Commodity Gap rSMCG as

$$\text{rSMCG} := \begin{cases} \frac{z^M - z^S}{|z^S|}, & \text{if } z^S \neq 0, \\ z^M, & \text{if } z^S = 0. \end{cases}$$

If $\mathcal{P}_M$ is infeasible while $\mathcal{P}_S$ is feasible, we define $\text{SMCG} = \text{rSMCG} := \infty$.

The fundamental property that the SMCG is non-negative is established in the following theorem.

Theorem 3.3. Let $\mathcal{P}_S = (V, E, u, w, b)$ be a feasible instance with optimal objective value z^S and let $\mathcal{P}_M = (V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$ where $u^k = u$ and $w^k = w$ for all $k \in K$, and $\{b^k\}_{k \in K}$ satisfies (3). If both problems $\mathcal{P}_S$ and $\mathcal{P}_M$ are feasible with optimal objective values z^S and z^M respectively, then $z^M \geq z^S$, i.e., $\text{SMCG} \geq 0$.

Proof. Let $\{f^k\}_{k \in K}$ be an optimal solution for $\mathcal{P}_M$. We define the aggregated flow vector $\bar{x} \in \mathbb{R}^{|E|}$ as $\bar{x}_e = \sum_{k \in K} f_e^k$ for all $e \in E$. Analogous to the proof of Proposition 3.1, $\bar{x}$ is a feasible solution for $\mathcal{P}_S$. The cost of this aggregated flow in $\mathcal{P}_S$ is:

$$\sum_{e \in E} w_e \bar{x}_e = \sum_{e \in E} w_e \left(\sum_{k \in K} f_e^k \right) = \sum_{k \in K} \sum_{e \in E} w_e f_e^k = z^M.$$

Since z^S is the optimal objective value of $\mathcal{P}_S$, and $\bar{x}$ is a feasible solution, we directly obtain that:

$$z^S \leq \sum_{e \in E} w_e \bar{x}_e.$$

Therefore, $z^S \leq z^M$. □

We remark that Theorem 3.3 ($z^M \geq z^S$) holds also if we relax the assumption that $u^k = u$ for all $k \in K$ to $u_e^k \in [0, u_e]$ for each $e \in E$ and $k \in K$. Under the assumption $u^k = u$ for all $k \in K$, the SMCG measures only the effect of demand partitioning rather than arbitrary capacity tightening.

4 Numerical Experiments

In this section, we evaluate the computational impact of transitioning from single-commodity to multicommodity flows using the meta-generation framework introduced in Section 2. The experiments are designed to investigate the effects of CDH, capacity restrictions, and problem size on instance tractability and solver performance.

4.1 Experimental Setup

All computational experiments were conducted on a workstation equipped with an Intel Core Ultra 7 255U processor and 32 GB of RAM. All solver runs were strictly restricted to a single thread. We benchmark the following solution methods, covering both commercial and open-source solvers.

We use the commercial solver **Gurobi 13.0.3** [28]:

- **Gurobi Primal:** Primal Simplex method (`Method = 0`).
- **Gurobi Dual:** Dual Simplex method (`Method = 1`).

- **Gurobi Barrier**: Barrier (Interior-Point) method without crossover (**Method** = 2, **Crossover** = 0).
- **Gurobi PDHG**: First-Order Primal-Dual Hybrid Gradient method without crossover (**Method** = 6, **Crossover** = 0).

We additionally use the open-source solver **HiGHS 1.15.1** [29]:

- **HiGHS Dual**: Dual Simplex method.
- **HiGHS IPM**: Interior-Point method without crossover.

As a decomposition-based alternative, we use the open-source solver **GCG 3.5.0** [6]:

- **GCG**: Dantzig-Wolfe decomposition using **PySCIP0pt 6.0.0** [30] with **SCIP 10.0.0** [31], and **PyGCG0pt 1.0.0** [6] interfaces. The standard MCMCF block-diagonal structure is manually supplied.

Models for Gurobi and HiGHS were formulated via **Pyomo 6.10.1** [32, 33]. All parameters were kept at the default values unless explicitly noted.

Primary evaluation metrics include runtime (in seconds) and the proportion of instances solved to optimality within the prescribed time limit. Additionally, we report the two structural metrics introduced in Sections 2.2 and 3: CDH and the (r)SMCG. To quantify network congestion and the degree to which commodities compete for shared capacity, we report the *active arc ratio* (AAR), defined as the fraction of bundle capacity constraints (2b) that are binding in the optimal solution.

4.1.1 Base Single-Commodity Instances

We generated four families of base single-commodity flow instances: **NETGEN** and **NETGEN-SR** using the **NETGEN** generator [18, 19], and **GRIDGEN** and **GRIDGEN-SR** using the **GridGen** generator (developed by Lee and Orlin [19, 20]). Each family contains five instances of increasing scale, denoted **<Family>-<i>** for $i \in \{1, \dots, 5\}$. We used the **NETGEN** and **GridGen** implementations provided by Frangioni [16].

All instances were generated using the parameters **cost_min** = 1, **cost_max** = 10000, **cap_min** = 1, and **cap_max** = 1000, following Kovács [34]. All base networks were initialized with the random seed 16042001. For **NETGEN** and **NETGEN-SR**, **transship_src** = 0 and **transship_trg** = 0 were enforced. For **GRIDGEN** and **GRIDGEN-SR**, two-way arcs (**2-way** = 1) and uniform distributions (**cost_dist** = 1, **cap_dist** = 1) were explicitly enforced.

The structural parameters for all base instances are summarized in Tables 1 and 2.

Table 1: Structural parameters for **NETGEN-<i>** and **NETGEN-SR-<i>** base instances.

Index (i)	n	m		src	trg	supply
		NETGEN-i	NETGEN-SR-i			
1	100	800	1,000	8	8	8,000
2	200	1,600	3,000	14	14	16,000
3	400	3,200	8,000	21	21	24,000
4	600	4,800	15,000	28	28	32,000
5	800	6,400	23,200	36	36	40,000

4.1.2 Multicommodity Instance Generation

The base single-commodity flow instances are transformed into Minimum-Cost Multicommodity Flow (MCMCF) instances via the meta-generation framework outlined in Section 2. All instances were generated using **s2mflow 0.3.1** [27] with the same fixed random seed of 16042001 as in Section 4.1.1.

No commodity-specific costs are considered, i.e., $w_e^k = w_e$ in all experiments. Unless explicitly varied in the capacity-tightening experiments, individual commodity capacities are set equal to their bundle

Table 2: Structural parameters for GRIDGEN- $\langle i \rangle$ and GRIDGEN-SR- $\langle i \rangle$ base instances.

Index (i)	n	width	src	trg	deg		supply
					GRIDGEN- i	GRIDGEN-SR- i	
1	100	8	8	8	8	8	8,000
2	200	14	14	14	8	14	16,000
3	400	21	21	21	8	21	24,000
4	600	28	28	28	8	28	32,000
5	800	36	36	36	8	36	40,000

capacity ($u_e^k = u_e$ for all $k \in \{1, \dots, K\}$ and $e \in E$). This isolates the computational impact of CDH, ensuring that algorithmic difficulty is not artificially dominated by individual capacity constraints.

To explore diverse demand structures, we apply four generation methods across all base instances:

- **Uniform (U):** We apply the *Uniform* partitioning method from Section 2.1.1.
- **Spread (S):** We apply the *Spread* partitioning method from Section 2.1.2.
- **Beta-Binomial ($B5$ and $B50$):** We apply the *Beta-Binomial* partitioning method from Section 2.1.3 with concentration parameters $c = 5$ and $c = 50$, respectively.

For each base instance $\langle \text{Family} \rangle\text{-}\langle i \rangle$, $i \in \{1, \dots, 5\}$, we generate multicommodity instances across commodity counts $K \in \{2, 3, 5, 10, 20, 50\}$. To investigate performance at large commodity counts in Section 4.2.3, instances derived from indices $i = 1, 2$ of each family are additionally generated for $K \in \{100, 200, 500, 1000\}$. An instance derived from base family $\langle \text{Family} \rangle$ index i , partitioning method $\langle \text{Method} \rangle$, and commodity count K is denoted by $\langle \text{Family} \rangle\text{-}i\text{-}\langle \text{Method} \rangle\text{-}K$. For example, NETGEN-SR-3-S-20 denotes the instance generated from index $i = 3$ of the NETGEN-SR family using the *Spread* method with $K = 20$ commodities.

To evaluate the impact of arc-capacity restrictions, we generate two classes of capacity-restricted variants:

- **Uniform Capacity Scaling (S):** For each arc $e \in E$ and commodity $k \in \{1, \dots, K\}$, individual capacities are sampled as $u_e^k \sim \mathcal{U}(\alpha \cdot u_e, u_e)$, with $\alpha = 0.5$. These instances are denoted by $\langle \text{Instance} \rangle\text{-}S$.
- **Zero-Capacity Restriction (Z):** To model commodity-specific arc exclusions, the density parameter $\rho = 0.05$ determines the fraction of restricted arcs per commodity. Specifically, for each commodity $k \in \{1, \dots, K\}$, we set $u_e^k = 0$ for $\tilde{m} = \max(1, \lfloor \rho |E| \rfloor)$ arcs selected at cyclic indices $((k \bmod |E|) + j \cdot \lceil |E| / \tilde{m} \rceil) \bmod |E|$ for $j = 0, \dots, \tilde{m} - 1$. These instances are denoted by $\langle \text{Instance} \rangle\text{-}Z$.

4.2 Results and Analysis

The computational study comprises five experiments. Experiment 1 characterizes how the Beta-Binomial concentration parameter controls CDH. Experiment 2 isolates the effect of CDH, and the induced SMCG, on solver performance, with $u_e^k = u_e$. Experiment 3 examines scaling behavior at large commodity counts. Experiments 4 and 5 analyze the impacts of tightened commodity-specific capacities and of zero-capacities.

4.2.1 Experiment 1: The Beta-Binomial Concentration Parameter Controls CDH

This experiment isolates the effect of the concentration parameter c on CDH. We computed CDH for $K = 10$ on base instance $\langle \text{Family} \rangle\text{-}3$ of all four families, using two additional random seeds (42 and 56789432) to average out sampling noise.

Figure 1 reports the mean CDH as a function of c . CDH decreases as c increases: larger values of c yield demands with lower CDH, whereas smaller values yield demands with higher CDH.

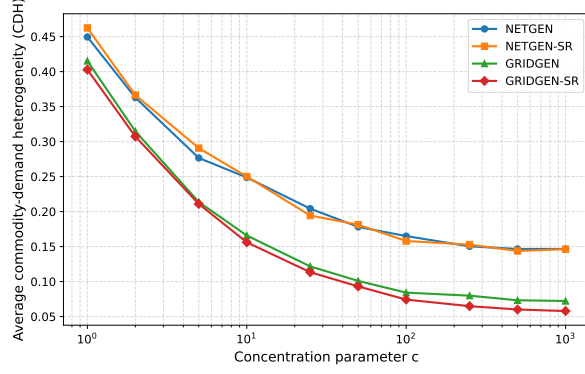


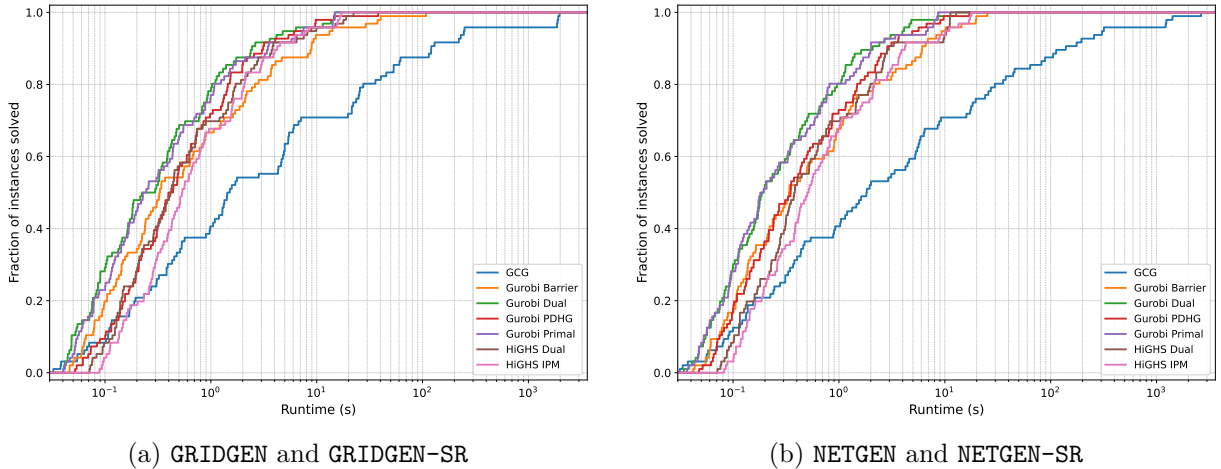
Figure 1: Effect of the Beta-Binomial concentration parameter c on CDH.

4.2.2 Experiment 2: Effect of Commodity-Demand Heterogeneity on Solver Performance

This experiment isolates the effect of CDH on solver performance, with individual commodity capacities set to the bundle capacity ($u_e^k = u_e$) so that no capacity restriction influences the comparison. We consider all base instances, commodity counts $K \in \{2, 3, 5, 10, 20, 50\}$, and a time limit of 3,600 s.

A preliminary experiment restricted to indices $i = 1, 2$ of all families and all partitioning methods showed that GCG requires significantly higher runtimes, already around 2,000 seconds (s) for instances of index 2 with $K = 50$; the index 3 instance **GRIDGEN-SR-3-B5-50** took around 18,000 s to solve.

Figure 2 compares GCG against all other methods on the reduced instance set. GCG is therefore excluded from the remaining experiments. We remark that Dantzig–Wolfe decomposition combined with column generation remains highly effective within branch-and-price frameworks for certain mixed-integer problems [5]. GCG provides a general-purpose solver based on this approach [6].



(a) **GRIDGEN** and **GRIDGEN-SR**

(b) **NETGEN** and **NETGEN-SR**

Figure 2: Runtime success profiles on instances with $i = 1, 2$ including **GCG**.

Among the remaining methods, **Gurobi Barrier** is the only one that fails to solve all instances within the time limit, timing out on a few of the largest instances ($i = 5$, $K = 50$). All other methods solve every test instance to optimality. **Gurobi Dual** is the fastest method on small and medium instances. **HiGHS IPM** is remarkably competitive: it solves all instances within the time limit and matches **Gurobi Dual** on the largest ones. Its behavior differs markedly from **Gurobi Barrier**, suggesting that the two interior-point implementations differ substantially in their numerical behavior. For **HiGHS** itself, **HiGHS Dual** is preferable for small K , while **HiGHS IPM** becomes preferable for $K \geq 20$. **Gurobi PDHG** follows a distinctive pattern: it is not competitive on small and medium instances but crosses every other method as instance size grows, becoming the fastest on the largest instances.

We note that **Gurobi PDHG** returns solutions with lower accuracy than the simplex and interior-point methods. In some cases, this yields marginally negative SMCG values, which reflects the imprecision of

the first-order method. **HiGHS IPM** and **Gurobi Barrier** produce solutions with precision close to that of the simplex methods.

Figure 3 reports runtime profiles across the four network families. The network topology has a distinct effect on solver behavior. On **GRIDGEN** and **NETGEN**, **HiGHS IPM** outperforms **Gurobi Primal** on large instances, while the two perform similarly on **GRIDGEN-SR**; on **NETGEN-SR**, **HiGHS IPM** is not better than **Gurobi Primal**. On the **NETGEN** families, the difference between **HiGHS Dual** and **Gurobi Barrier** is small; on **GRIDGEN** and especially **GRIDGEN-SR**, it is pronounced in favor of **HiGHS Dual**. The gap between **Gurobi PDHG** and **Gurobi Dual** on large instances is widest on **GRIDGEN** and **GRIDGEN-SR** and is smaller on the **NETGEN** families. Finally, on the **SR** families, the gap between **Gurobi Dual** and **Gurobi Primal** is smaller than on the non-SR families, where **Gurobi Dual** retains the advantage.

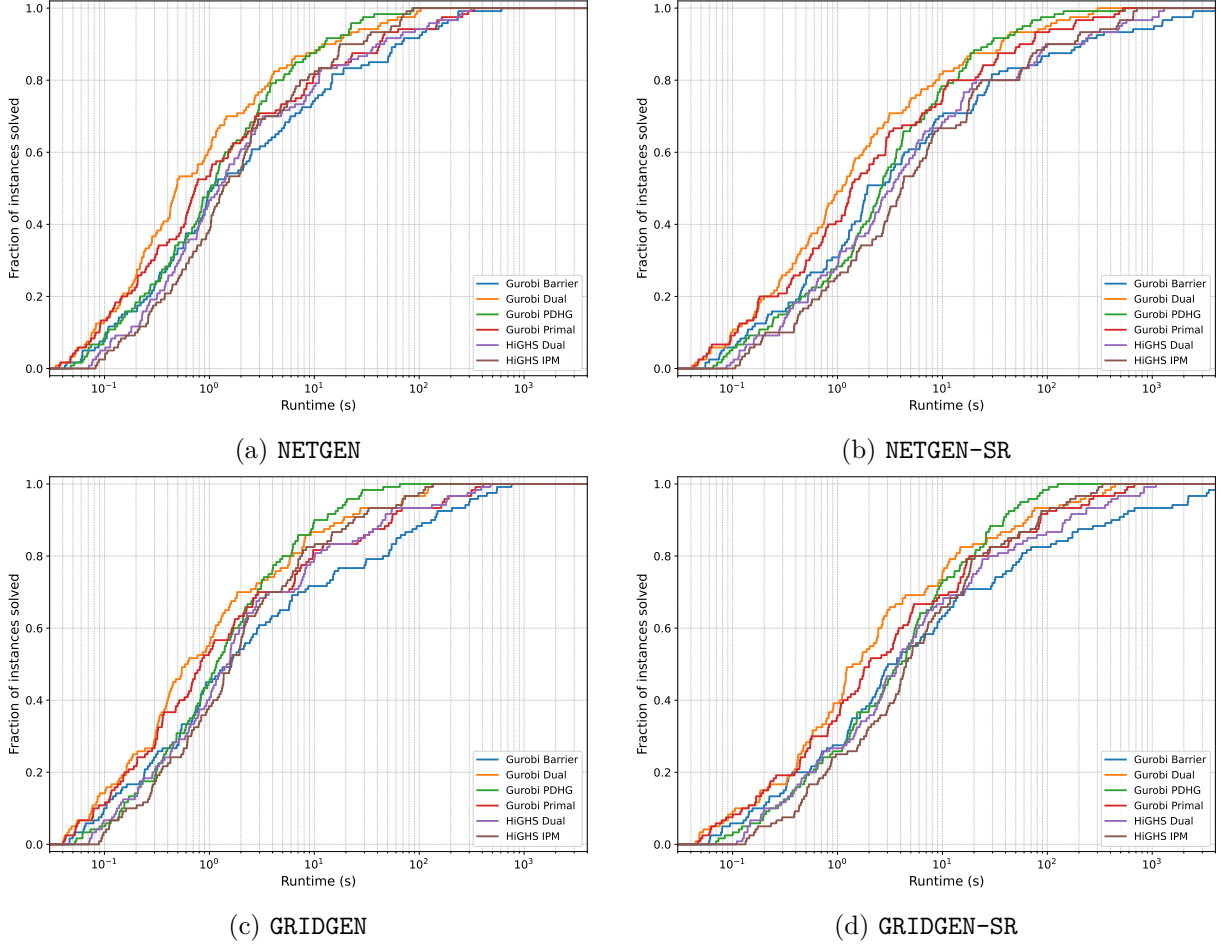


Figure 3: Runtime success profiles per network family.

Figure 4 reports runtime profiles across the four integer partitioning methods, ordered by increasing CDH: *Uniform* (low), *B50* (low-to-moderate), *B5* (moderate), and *Spread* (high). The four panels share the same qualitative shape, and the pairwise ordering of methods is largely stable across them. Two relations change direction between the low-CDH and high-CDH regimes. **HiGHS IPM** performs similarly to **Gurobi Primal** on large instances under *Uniform*, but outperforms **Gurobi Primal** under *B50*, *B5*, and *Spread* on large instances. **Gurobi PDHG** is the fastest method on large instances under *Uniform*, *B50*, and *B5*, but **Gurobi Dual** is fastest under *Spread*, with similar runtimes to **Gurobi PDHG** on large instances. The *Spread* panel is uniformly flatter, meaning that runtimes increase with higher CDH. **Gurobi Barrier** is also comparatively slower under *B5* and *Spread* than under *Uniform* and *B50*. The differences between the four panels are moderate: the ranking of the best methods changes under high CDH, and the runtimes shift uniformly upward with increasing CDH, but the underlying ordering of the remaining methods is largely preserved. This is in contrast to the effects observed in Sections 4.2.3, 4.2.4, and 4.2.5, where the differences in ranking and runtime are substantially larger.

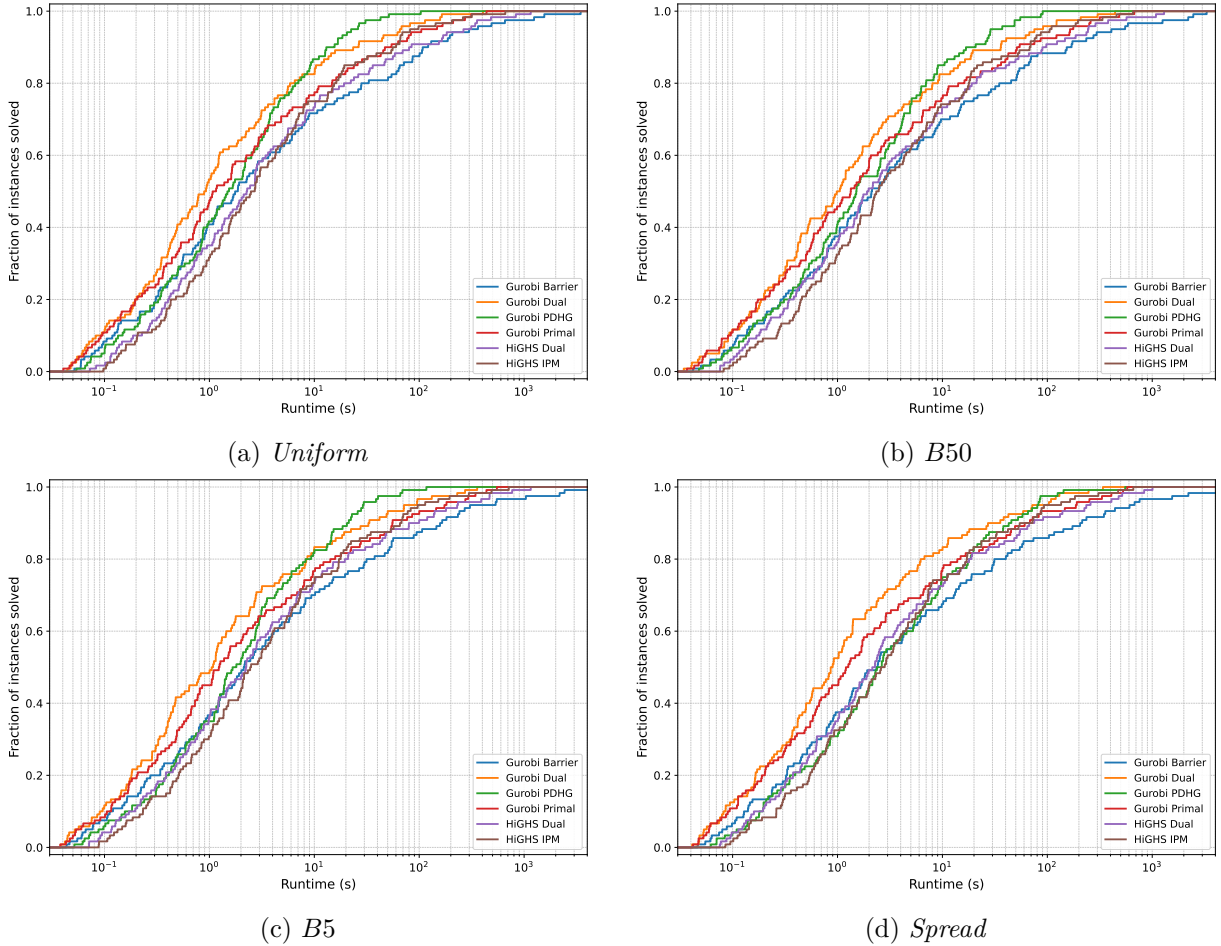


Figure 4: Runtime success profiles per integer partitioning method.

The CDH induced by the integer partitioning method has a clear effect on the Single-Multi-Commodity Gap. As reported in Table 3, *Uniform* partitioning yields instances with $\text{SMCG} \approx 0$, suggesting that the aggregated single-commodity flow acts as a strong relaxation, whereas *Spread* partitioning induces a positive rSMCG of approximately 0.01. The active arc ratio (AAR) shows a small decrease across all partitioning methods relative to the base value, indicating that moving to multicommodity flow instances tends to slightly reduce the fraction of bundle constraints that are binding at the optimum.

4.2.3 Experiment 3: Scaling to Large Commodity Counts

This experiment examines solver behavior as the number of commodities grows to $K \in \{100, 200, 500, 1000\}$, and includes the smaller values $K \in \{10, 20, 50\}$. We restrict the experiment to instances derived from indices $i = 1, 2$ of each family, and continue to use $u_e^k = u_e$, so that the effect of scale is studied in isolation from capacity effects. We apply a time limit of 1,200 s. Figure 5 reports the runtime profiles, split into *Uniform* and non-*Uniform* panels, where the latter aggregates the *B50*, *B5*, and *Spread* partitioning methods.

As is well known, runtimes increase substantially as K grows; we observe the same here. The ordering of methods changes relative to Experiment 2, with the reordering driven by the CDH induced by the integer partitioning method. **Gurobi Dual** and **Gurobi Primal** perform worst of the benchmarked solvers on the non-*Uniform* instances. On *Uniform* instances, **Gurobi Primal** is slightly better than **Gurobi Dual**, contrary to the pattern observed at moderate K in Section 4.2.2. **Gurobi PDHG** consolidates its advantage and becomes the fastest method on the non-*Uniform* instances. **Gurobi Barrier**, which was the weakest method up to $K = 50$ in Experiment 2, solves all instances within the time limit and is, together with **HiGHS IPM**, among the fastest methods on non-*Uniform* instances.

The two panels differ. In the *Uniform* panel, the curves separate: **Gurobi PDHG**, **Gurobi Barrier**,

Table 3: Statistics on the generated instances in Experiment 2. Values are means across all families and indices. CDH is averaged over all instances; rSMCG, SMCG, and AAR over feasible instances only. The base single-commodity instances, solved with **Gurobi Dual**, have an average AAR of 0.0264; their objective values are used to compute the SMCG and rSMCG.

Strategy	Metric	K=2	K=3	K=5	K=10	K=20	K=50
<i>Uniform</i>	CDH	0.0051	0.0082	0.0108	0.0181	0.0267	0.0491
	rSMCG	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	SMCG	0	0	0	0	0	0
	AAR	0.0246	0.0247	0.0246	0.0248	0.0245	0.0244
<i>B50</i>	CDH	0.1126	0.1141	0.1062	0.1121	0.1264	0.1632
	rSMCG	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	SMCG	0	8	11	95	258	2,296
	AAR	0.0248	0.0246	0.0246	0.0247	0.0246	0.0244
<i>B5</i>	CDH	0.3030	0.2662	0.2355	0.2285	0.2334	0.2524
	rSMCG	0.0007	0.0007	0.0002	0.0006	0.0008	0.0008
	SMCG	59,584	62,332	28,390	62,655	70,797	94,197
	AAR	0.0249	0.0248	0.0246	0.0245	0.0247	0.0243
<i>Spread</i>	CDH	0.5517	0.4490	0.4323	0.4181	0.4108	0.4067
	rSMCG	0.0105	0.0105	0.0096	0.0111	0.0129	0.0110
	SMCG	1,760,312	1,639,434	1,437,810	1,596,721	1,704,974	1,477,162
	AAR	0.0259	0.0251	0.0247	0.0245	0.0249	0.0241
Instances		80	80	80	80	80	80
Infeasible		0	0	0	0	0	0

and **Gurobi Primal** lead, followed by **Gurobi Dual** and **HiGHS IPM**. **HiGHS Dual** performs worst on *Uniform* instances and does not solve all of them within the time limit. In the non-*Uniform* panel, the curves cluster more tightly: **Gurobi PDHG**, **Gurobi Barrier**, and **HiGHS IPM** lead, while **Gurobi Dual**, **Gurobi Primal**, and **HiGHS Dual** are close to each other. **HiGHS Dual** performs slightly better than the **Gurobi** simplex methods on large non-*Uniform* instances. On the *Uniform* instances, both **HiGHS** methods perform visibly worse than the **Gurobi** methods, for which one reason could be a more aggressive presolve in **Gurobi** recognizing and exploiting the homogeneous commodity-demand structure of these instances, as presolve can significantly accelerate the solution process [35].

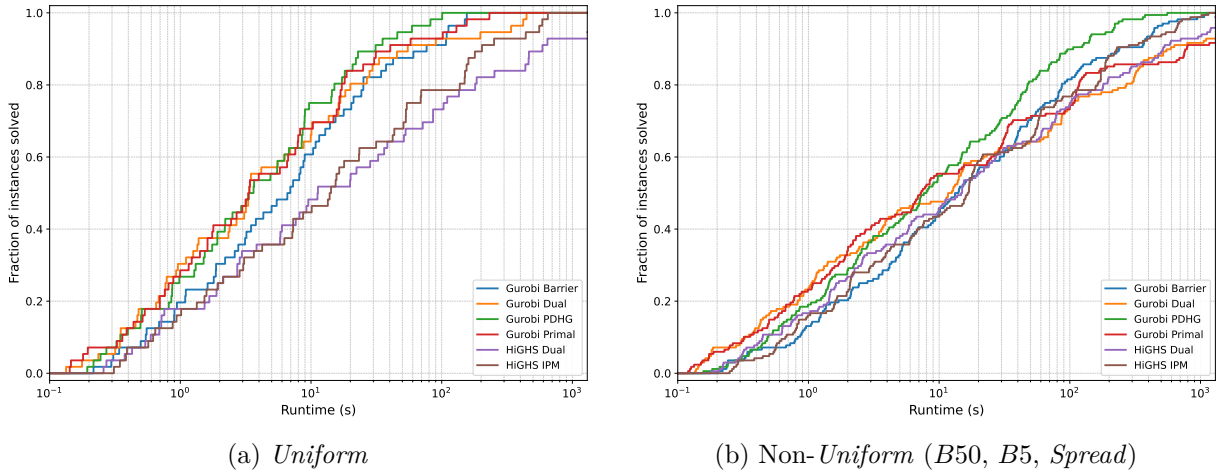


Figure 5: Runtime scalability on large commodity sets.

Table 4 reports the aggregated statistics. CDH grows with K for every partitioning method. The CDH for *Uniform* also grows substantially with K . This is due to the combined effect of a smaller ratio of total supply to the number of commodities and of the global balancing phase: under a fully uniform distribution, the per-commodity supply is small for large K , so that small integer deviations already

increase CDH substantially. The rSMCG follows the same direction: it is zero for *Uniform* at all K and grows monotonically for the other methods, reaching 0.023 for *Spread* at $K = 1000$. The AAR is stable across K for all strategies, and slightly below the base value of 0.0338.

Table 4: Aggregated gap and heterogeneity statistics for large commodity counts. Values are means across all families and indices. CDH is averaged over all instances; rSMCG, SMCG, and AAR over feasible instances only. The base single-commodity instances, solved with **Gurobi Dual**, have an average AAR of 0.0338; their objective values are used to compute the SMCG and rSMCG.

Strategy	Metric	K=100	K=200	K=500	K=1000
<i>Uniform</i>	CDH	0.0722	0.1144	0.2199	0.3280
	rSMCG	0.0000	0.0000	0.0009	0.0027
	SMCG	0	0	73,820	208,955
	AAR	0.0318	0.0316	0.0313	0.0309
<i>B50</i>	CDH	0.2046	0.2652	0.3820	0.4943
	rSMCG	0.0001	0.0014	0.0066	0.0160
	SMCG	4,278	104,468	501,116	1,259,628
	AAR	0.0323	0.0312	0.0317	0.0305
<i>B5</i>	CDH	0.2778	0.3253	0.4187	0.5170
	rSMCG	0.0027	0.0048	0.0098	0.0200
	SMCG	197,551	362,468	756,585	1,596,162
	AAR	0.0313	0.0317	0.0320	0.0309
<i>Spread</i>	CDH	0.4116	0.4236	0.4676	0.5201
	rSMCG	0.0170	0.0193	0.0200	0.0230
	SMCG	1,349,131	1,527,656	1,600,335	1,860,576
	AAR	0.0314	0.0312	0.0318	0.0303
Instances		32	32	32	32
Infeasible		0	0	0	0

The strong performance of **HiGHS IPM** and **Gurobi Barrier** at large K shows that interior-point methods are highly relevant for MCMCF. For general interior-point methods, the computation of the Newton direction becomes expensive due to the block-angular structure of the constraint matrix [11]. Specialized interior-point methods for MCMCF were developed precisely to exploit this structure and are, in principle, superior to general-purpose implementations [11]. At the same time, general-purpose dual simplex methods are on some instances competitive with the specialized interior-point methods [26]. The observed results support both perspectives: general-purpose interior-point methods are highly competitive on large- K instances, which supports the case for the specialized interior-point methods of [11, 36]. For moderate K , the dual simplex method remains very competitive.

4.2.4 Experiment 4: Uniform Capacity Scaling (S)

This experiment introduces commodity-specific capacities that are tighter than the bundle capacity. For each arc $e \in E$ and commodity $k \in \{1, \dots, K\}$, capacities are sampled as $u_e^k \sim \mathcal{U}(\alpha \cdot u_e, u_e)$ with $\alpha = 0.5$. All base instances are used, and the time limit remains 1,200 s. Figure 6 reports runtime profiles per network family, and Figure 7 per integer partitioning method.

The ranking of methods changes compared to Experiment 2. **HiGHS IPM** is now highly competitive on large instances, outperforming **Gurobi PDHG** and **Gurobi Dual** on some. On the **NETGEN** families it is indistinguishable from **Gurobi PDHG** on large instances, whereas **Gurobi PDHG** remains slightly better on the **GridGen** families. On the largest instances, **Gurobi Primal** and **HiGHS Dual** are nearly indistinguishable. The performance difference between **Gurobi Dual** and **Gurobi Primal** remains, with **Gurobi Dual** performing better. **Gurobi Barrier** remains the slowest method, with a larger difference to the other methods on **NETGEN-SR**, more so than in Experiment 2. On small and medium instances, **Gurobi Dual** and **Gurobi Primal** perform best.

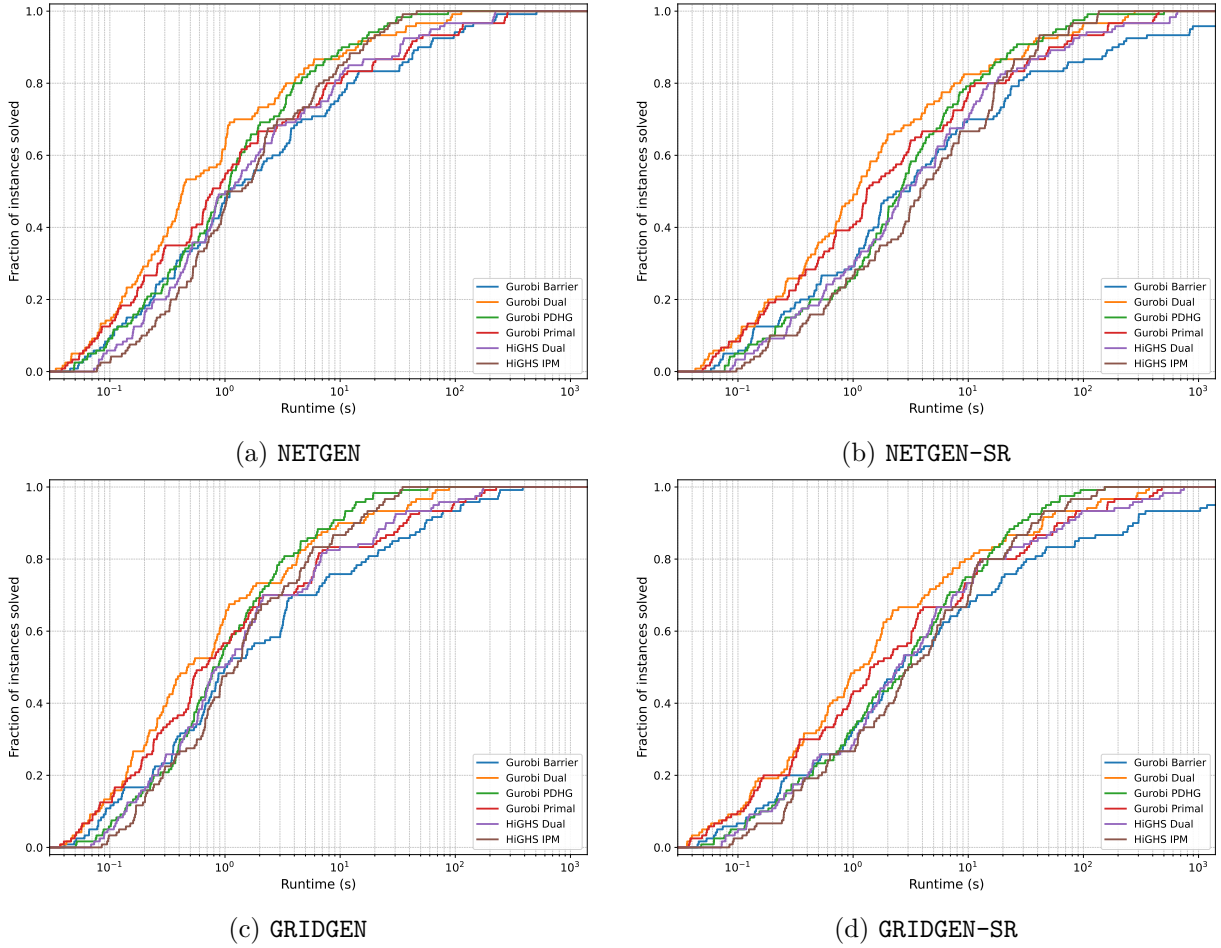


Figure 6: Runtime profiles under uniform capacity scaling ($\alpha = 0.5$), per network family.

At the partitioning level, the CDH effect observed in Experiment 2 persists: for higher CDH (*B5* and *Spread*), runtimes increase. However, the relative performance of the solvers shifts. In particular under *Spread*, HiGHS IPM is very successful on large instances, so that the advantage of Gurobi PDHG becomes clearly smaller; on the largest *Spread* instances, HiGHS IPM is even slightly superior. The relative behavior between Gurobi PDHG and Gurobi Dual remains unchanged.

Table 5 reports the aggregated statistics. The rSMCG increases for the higher-CDH partitioning methods relative to Experiment 2, and remains near zero for the others. The AAR is essentially unchanged across all strategies; only for the higher-CDH instances (*B5* and *Spread*) and small K values is it slightly decreased relative to Experiment 2.

4.2.5 Experiment 5: Zero-Capacity Restriction (Z)

This experiment models commodity-specific arc exclusions. For each commodity, a fraction $\rho = 0.05$ of arcs receives $u_e^k = 0$, following the deterministic round-robin scheme from Section 4.1.2. All base instances are used, and the time limit is 1,200 s. Under this generation scheme, multiple instances become infeasible, as reported in Table 6.

Figure 8 reports runtime profiles per network family. The qualitative patterns from Experiment 4 largely persist under zero capacities. Gurobi Barrier remains the slowest method, and HiGHS IPM remains highly competitive on large instances. Gurobi Dual is again among the best methods.

Figure 9 reports runtime profiles per integer partitioning method. Gurobi Primal performs worse than in Experiment 2 on large instances. Gurobi PDHG retains its advantage over Gurobi Dual on large instances. Overall, for higher-CDH instances (*B50*, *B5*, *Spread*), HiGHS IPM is very competitive and, on some large instances, is the best method, outperforming Gurobi PDHG.

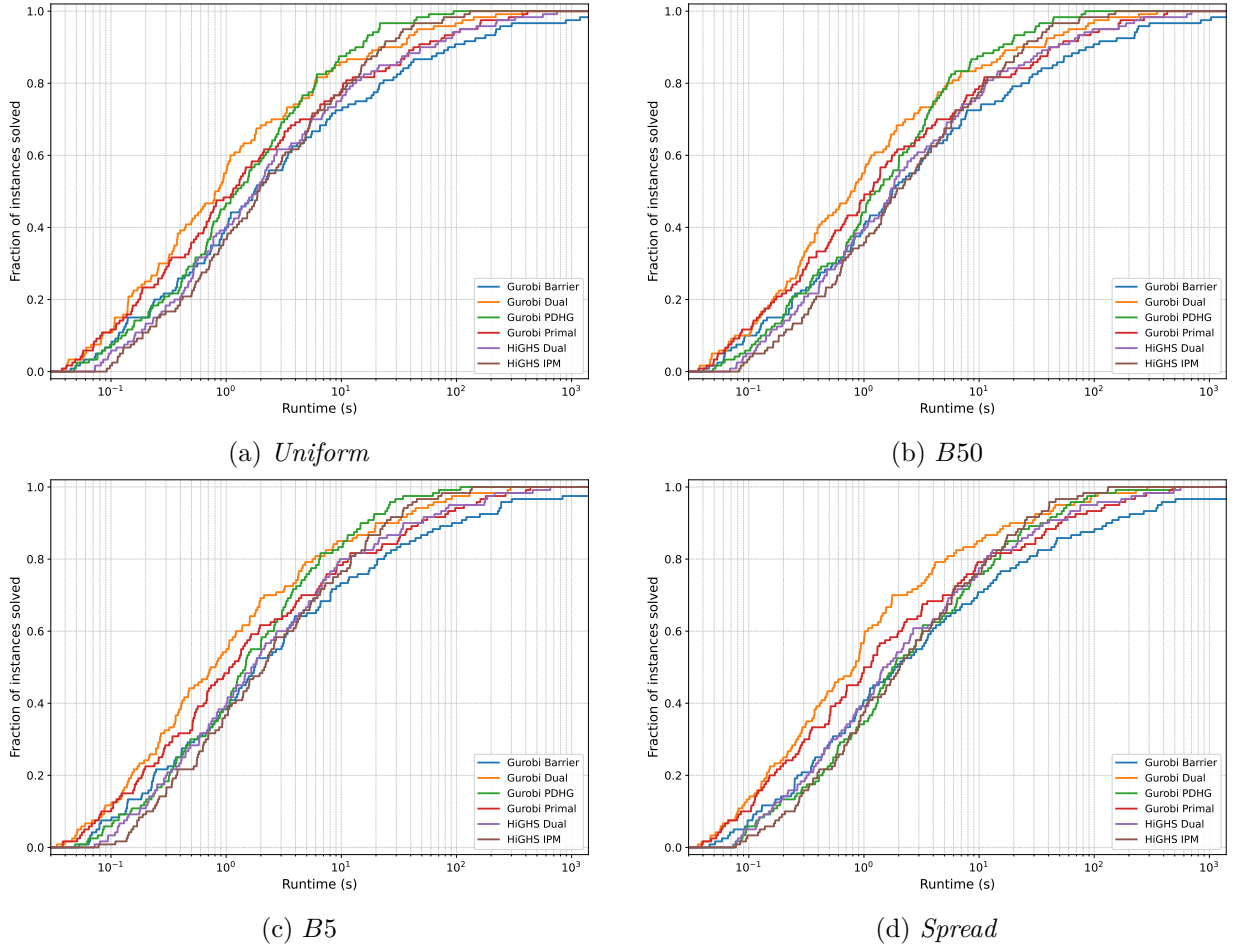


Figure 7: Runtime profiles under uniform capacity scaling with $\alpha = 0.5$, per integer partitioning method.

Table 5: Statistics with capacity scaling ($\alpha = 0.5$). The base instances and demand vectors are identical to those in Experiment 2, so CDH values equal those reported in Table 3. Values are means across all families and indices, computed over feasible instances only. The base single-commodity instances, solved with **Gurobi Dual**, have an average AAR of 0.0264; their objective values are used to compute the rSMCG.

Strategy	Metric	K=2	K=3	K=5	K=10	K=20	K=50
Uniform	rSMCG	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	SMCG	0	0	0	0	0	0
	AAR	0.0248	0.0246	0.0247	0.0246	0.0245	0.0245
<i>B50</i>	rSMCG	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	SMCG	719	723	11	95	258	2,296
	AAR	0.0248	0.0247	0.0246	0.0248	0.0245	0.0243
<i>B5</i>	rSMCG	0.0040	0.0011	0.0002	0.0006	0.0008	0.0008
	SMCG	673,312	106,306	30,311	62,655	70,800	94,197
	AAR	0.0239	0.0246	0.0246	0.0245	0.0246	0.0244
Spread	rSMCG	0.0482	0.0212	0.0135	0.0115	0.0130	0.0110
	SMCG	8,492,028	3,720,177	2,240,167	1,679,745	1,721,100	1,479,571
	AAR	0.0172	0.0223	0.0241	0.0242	0.0250	0.0242
Instances		80	80	80	80	80	80
Infeasible		0	0	0	0	0	0

Table 6 reports the aggregated statistics. The rSMCG increases substantially relative to Experi-

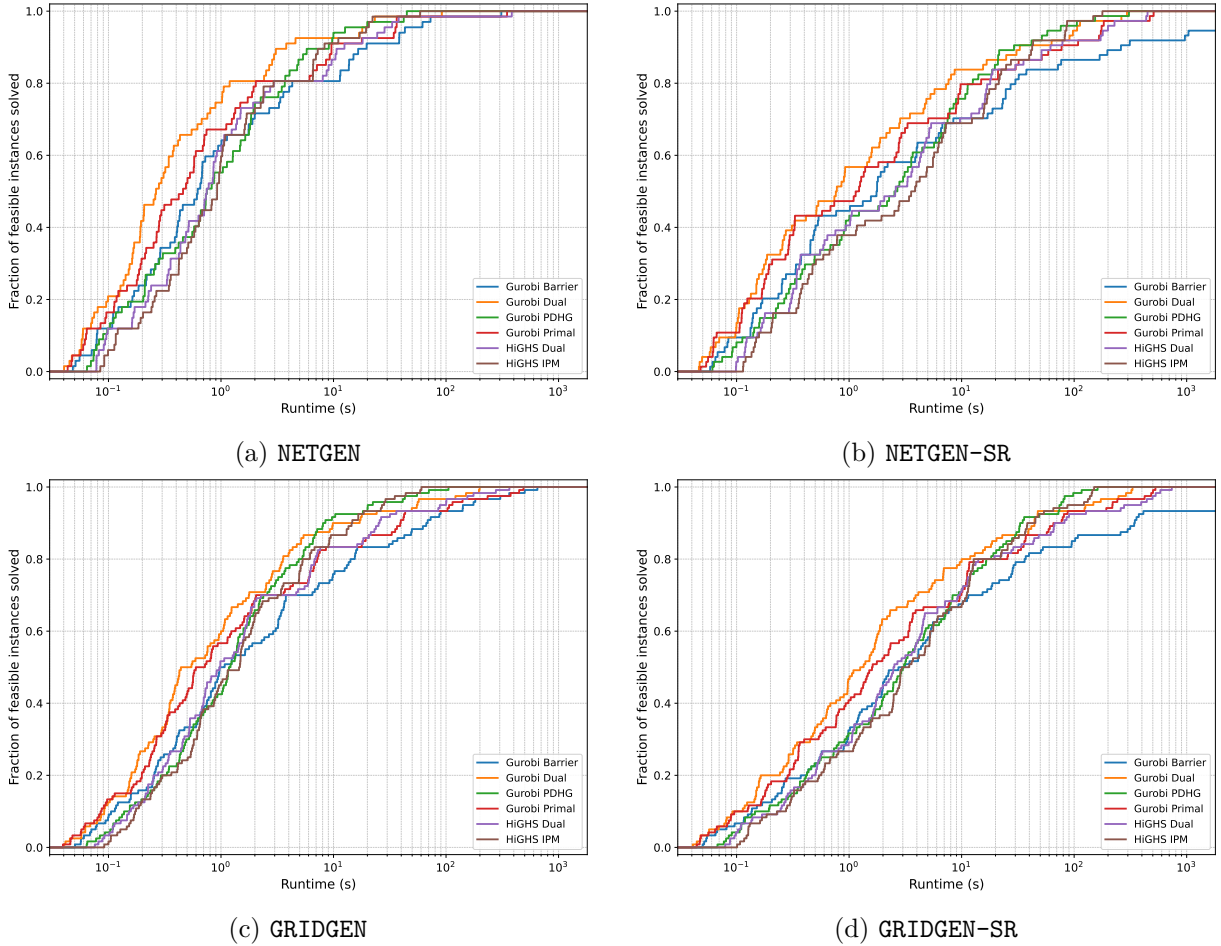


Figure 8: Runtime profiles under zero-capacity restrictions ($\rho = 0.05$), per network family.

ment 2: now, also the *Uniform* and *B50* instances exhibit a positive rSMCG. The rSMCG also increases for *B5* and *Spread* across all commodity counts. At the same time, we note that infeasible instances now occur, so the values are not entirely comparable. The AAR remains largely unchanged, with a slight tendency to be higher, which is consistent with the expected behavior.

Zero capacities substantially increase the rSMCG across all partitioning methods, including *Uniform* and *B50*, where the average rSMCG was zero without capacity restrictions. The AAR remains largely unchanged. This confirms that commodity-specific capacity restrictions can produce a positive SMCG even for low-CDH instances.

4.3 Interpretation of the Computational Experiments

Taken together, the experiments suggest that, alongside the network structure and the number of commodities, CDH has a substantial influence on the behavior of solution methods for MCMCF instances. CDH can be controlled through the meta-generation framework presented in this work. The experiments further suggest that simplex methods (**Gurobi Dual** and **Gurobi Primal**) perform best on small and medium instances. For large instances with many commodities, interior-point methods are superior, and the GPU-friendly first-order method **Gurobi PDHG** also performs well. At the same time, we observe that **Gurobi Dual** remains highly competitive up to a moderate commodity count of 50. The results of **Gurobi PDHG** are promising, though the method produces solutions with substantially lower accuracy than interior-point methods. We conclude that CDH should be treated as an important feature for the documentation and classification of benchmark instances. It can also serve as an explanatory feature in future machine-learning-based performance prediction methods for solution algorithms on MCMCF instances. Altogether, to generate hard instances, the meta-generation framework should be applied

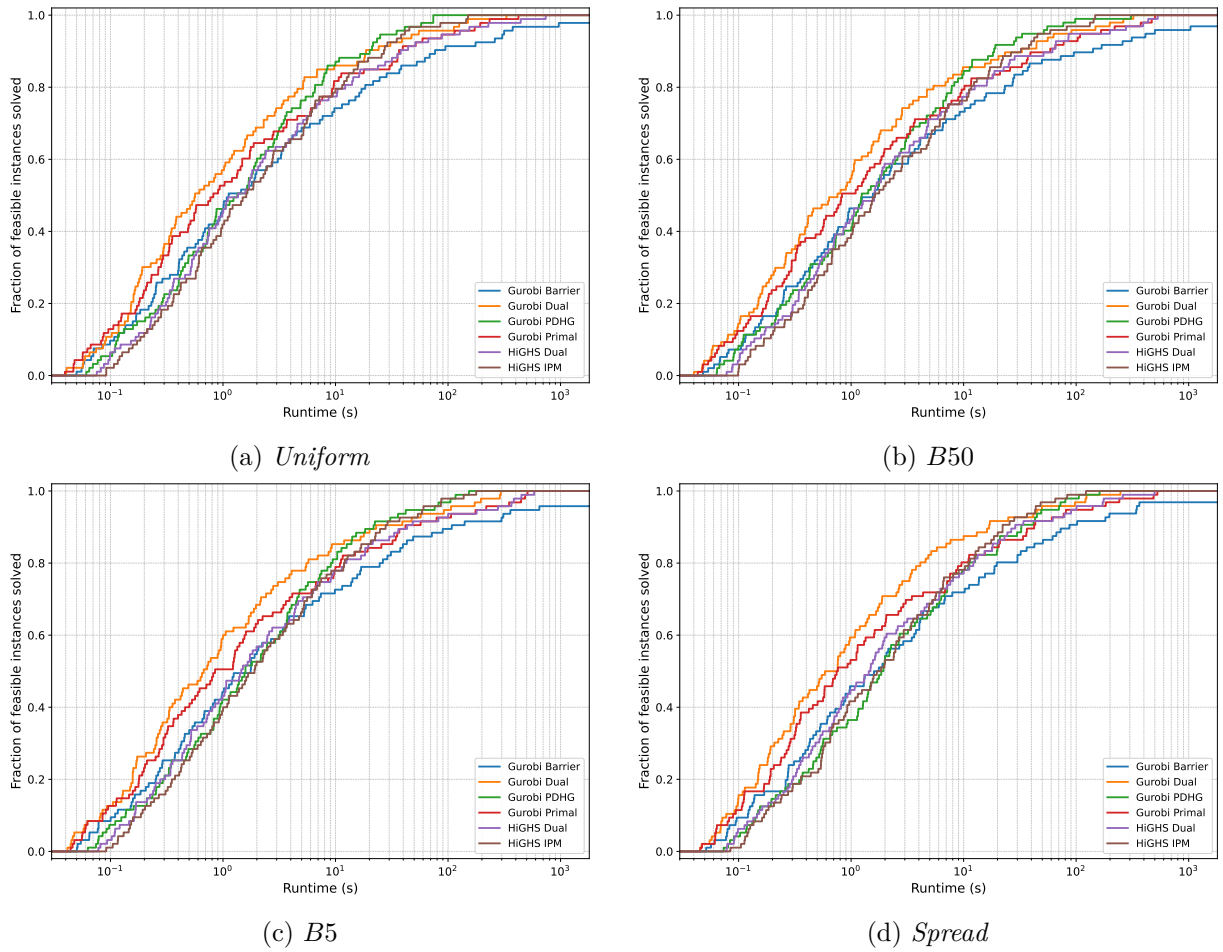


Figure 9: Algorithmic performance under zero-capacity restrictions ($\rho = 0.05$), per integer partitioning method.

Table 6: Statistics with zero capacities at rate $\rho = 0.05$. The base instances and demand vectors are identical to those in Experiment 2, so CDH values equal those reported in Table 3. Values are means across all families and indices, computed over feasible instances only. The base single-commodity instances, solved with **Gurobi Dual**, have an average AAR of 0.0264; their objective values are used to compute the SMCG and rSMCG.

Strategy	Metric	K=2	K=3	K=5	K=10	K=20	K=50
Uniform	rSMCG	0.0076	0.0037	0.0020	0.0015	0.0010	0.0010
	SMCG	1,523,552	577,696	402,348	284,389	157,911	154,955
	AAR	0.0259	0.0262	0.0256	0.0241	0.0227	0.0230
B50	rSMCG	0.0082	0.0041	0.0022	0.0017	0.0012	0.0013
	SMCG	1,569,723	626,201	412,327	293,991	164,517	193,558
	AAR	0.0250	0.0255	0.0249	0.0231	0.0218	0.0221
B5	rSMCG	0.0130	0.0064	0.0034	0.0023	0.0025	0.0026
	SMCG	2,413,866	1,083,428	683,846	399,599	316,272	449,041
	AAR	0.0250	0.0259	0.0256	0.0239	0.0228	0.0229
Spread	rSMCG	0.0367	0.0299	0.0214	0.0201	0.0193	0.0196
	SMCG	7,059,260	4,607,695	3,564,298	3,648,548	3,322,676	2,634,685
	AAR	0.0273	0.0274	0.0264	0.0232	0.0244	0.0244
Instances		80	80	80	80	80	80
Infeasible		2	11	15	18	27	26

using the *Spread* method or the *Beta-Binomial* method with a low concentration parameter, optionally with additional commodity-specific capacities.

5 Conclusions

We introduced a systematic meta-generation framework that lifts single-commodity minimum-cost flow instances into the multicommodity space while strictly preserving network topology and nodal roles. To make the resulting demand structure measurable and controllable, we defined commodity-demand heterogeneity (CDH) and introduced three integer partitioning methods: *Uniform* yields homogeneous demand, *Spread* yields highly heterogeneous demand, and *Beta-Binomial* provides control over CDH through its concentration parameter. We further established the *(relative) Single-Multi-Commodity Gap* (r)SMCG as a formal metric for the divergence in optimal objective values induced by the lifting process. We showed that CDH drives the SMCG and systematically influences solver behavior. Together, these results establish CDH as a relevant structural property of MCMCF instances. We conclude that CDH should be treated as an important feature for the documentation and classification of benchmark instances. It can also serve as an explanatory feature in future machine-learning-based performance prediction methods for solution algorithms on MCMCF instances. By releasing the open-source Python library `s2mflow`, we provide the community with a tool for generating reproducible, high-fidelity benchmark instances.

Acknowledgements

This work was supported by the Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR, German Federal Ministry of Research, Technology and Space) [*QuSol: Quantum Optimization Solver Kit*].

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Declaration of generative AI use

During the preparation of this work, the authors used Gemini (Google), DeepL Write, and DeepSeek in order to improve the readability and language of the manuscript. After using these services, the authors carefully reviewed, revised, and edited the content as needed. The authors take full responsibility for the content of the published article.

Author contributions: CRediT

F.P.B.: Conceptualization, Data curation, Formal Analysis, Investigation, Methodology, Project Administration, Software, Validation, Writing – original draft, Writing – review and editing. S.N.: Conceptualization, Funding Acquisition, Project Administration, Supervision, Writing – review and editing.

References

- [1] L. R. Ford, D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
- [2] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, NJ, 1993.

- [3] L. R. Ford, D. R. Fulkerson, A suggested computation for maximal multi-commodity network flows, *Management Science* 5 (1) (1958) 97–101.
URL <http://www.jstor.org/stable/2626975>
- [4] G. B. Dantzig, P. Wolfe, Decomposition principle for linear programs, *Operations Research* 8 (1) (1960) 101–111.
URL <http://www.jstor.org/stable/167547>
- [5] J. Desrosiers, M. Lübbecke, G. Desaulniers, J. B. Gauthier, *Branch-and-Price*, Springer Nature Switzerland, Cham, 2026. doi:10.1007/978-3-031-96917-1.
- [6] G. Gamrath, M. E. Lübbecke, Experiments with a generic dantzig-wolfe decomposition for integer programs, in: P. Festa (Ed.), *Experimental Algorithms*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 239–252. doi:10.1007/978-3-642-13193-6_21.
- [7] F. Vanderbeck, G. Marques, R. Sadykov, V. Nesello, N. Javerzat, A. Pessoa, T. Bulhões, C. Bentes, V. L. de Lima, A. Subramanian, Coluna.jl: A branch-cut-and-price framework in julia, *INFORMS Journal on Computing* (2026). doi:10.1287/ijoc.2025.1130.
- [8] A. A. Assad, Multicommodity network flows—a survey, *Networks* 8 (1) (1978) 37–91. doi:<https://doi.org/10.1002/net.3230080107>.
- [9] J. L. Kennington, A survey of linear cost multicommodity network flows, *Operations Research* 26 (2) (1978) 209–236.
URL <http://www.jstor.org/stable/169631>
- [10] P. Cappanera, A. Frangioni, Symmetric and asymmetric parallelization of a cost-decomposition algorithm for multicommodity flow problems, *INFORMS Journal on Computing* 15 (4) (2003) 369–384. doi:10.1287/ijoc.15.4.369.24887.
- [11] J. Castro, A specialized interior-point algorithm for multicommodity network flows, *SIAM Journal on Optimization* 10 (3) (2000) 852–877. doi:10.1137/S1052623498341879.
- [12] J. Castro, N. Nabona, An implementation of linear and nonlinear multicommodity network flows, *European Journal of Operational Research* 92 (1) (1996) 37–53. doi:[https://doi.org/10.1016/0377-2217\(95\)00137-9](https://doi.org/10.1016/0377-2217(95)00137-9).
- [13] F. Zhang, S. Boyd, Solving large multicommodity network flow problems on gpus (2025). *arXiv:2501.17996*.
URL <https://arxiv.org/abs/2501.17996>
- [14] A. Itai, Two-commodity flow, *J. ACM* 25 (1978) 596–611. doi:10.1145/322092.322100.
- [15] M. Ding, R. Kyng, P. Zhang, Two-Commodity Flow Is Equivalent to Linear Programming Under Nearly-Linear Time Reductions, in: M. Bojańczyk, E. Merelli, D. P. Woodruff (Eds.), *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, Vol. 229 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2022, pp. 54:1–54:19. doi:10.4230/LIPIcs.ICALP.2022.54.
- [16] A. Frangioni, Commalab: Datasets, single-commodity min-cost flow problems (2026).
URL <https://commalab.di.unipi.it/datasets/mcf/>
- [17] A. Frangioni, Commalab: Datasets, multicommodity flow problems (2026).
URL <https://commalab.di.unipi.it/datasets/mmcf/>
- [18] D. Klingman, A. Napier, J. Stutz, Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems, *Management Science* 20 (5) (1974) 814–821. doi:10.1287/mnsc.20.5.814.

- [19] DIMACS, Network flows and matching: The first dimacs implementation challenge: 1990-1991 (1990 – 1991).
URL <https://archive.dimacs.rutgers.edu/archive/pub/netflow/>
- [20] Y. Lee, Computational analysis of network optimization algorithms, Ph.d. dissertation, Massachusetts Institute of Technology, Cambridge, MA, department of Civil and Environmental Engineering. Supervisor: J. B. Orlin (1993).
- [21] DIMACS, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science (2026).
URL <http://dimacs.rutgers.edu>
- [22] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, S. J. Wichmann, An empirical evaluation of the korbx algorithms for military airlift applications, *Operations Research* 38 (2) (1990) 240–248.
URL <http://www.jstor.org/stable/171334>
- [23] A. Ali, J. L. Kennington, Mnetgen program documentation, Tech. Rep. Technical Report 77003, Department of Industrial Engineering and Operations Research, Southern Methodist University, Dallas, TX (1977).
- [24] A. Frangioni, G. Gallo, A bundle type dual-ascent approach to linear multicommodity min-cost flow problems, *INFORMS Journal on Computing* 11 (4) (1999) 370–393. doi:10.1287/ijoc.11.4.370.
- [25] J. Castro, N. Nabona, Computational tests of a linear multicommodity network flow code with linear side constraints through primal partitioning, Research Report DR 94/02, Statistics & Operations Research Dept., Universitat Politècnica de Catalunya (Feb. 1994).
URL <https://www-eio.upc.edu/~jcastro/publications/reports/dr1994-02.pdf>
- [26] R. E. Bixby, Solving real-world linear programs: A decade and more of progress, *Operations Research* 50 (1) (2002) 3–15. doi:10.1287/opre.50.1.3.17780.
- [27] F. P. Broesamle, S. Nickel, s2mflow: A meta-generator for multicommodity flow instances, *SoftwareX* 35 (2026) 102874. doi:<https://doi.org/10.1016/j.softx.2026.102874>.
- [28] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2026).
URL <https://www.gurobi.com>
- [29] Q. Huangfu, J. A. J. Hall, Parallelizing the dual revised simplex method, *Mathematical Programming Computation* 10 (1) (2018) 119–142. doi:10.1007/s12532-017-0130-5.
- [30] S. Maher, M. Miltenberger, J. P. Pedroso, D. Rehfeldt, R. Schwarz, F. Serrano, Pyscipopt: Mathematical programming in python with the scip optimization suite, in: G.-M. Greuel, T. Koch, P. Paule, A. Sommesse (Eds.), *Mathematical Software – ICMS 2016*, Springer International Publishing, Cham, 2016, pp. 301–307.
- [31] C. Hojny, M. Besançon, K. Bestuzheva, S. Borst, J. Dionísio, J. Ehls, L. Eifler, M. Ghannam, A. Gleixner, A. Göß, A. Hoen, J. von Holly-Ponientzietz, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, M. Lübbecke, S. J. Maher, P. M. Meinhold, G. Mexi, T. Mohr, E. Mühmer, K. K. Patel, M. E. Pfetsch, S. Pokutta, C. R. Groba, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, M. Walter, D. Weninger, L. Xu, The scip optimization suite 10.0 (2025). [arXiv:2511.18580](https://arxiv.org/abs/2511.18580).
- [32] M. L. Bynum, G. A. Hackebeit, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Sirola, J.-P. Watson, D. L. Woodruff, *Pyomo—optimization modeling in python*, 3rd Edition, Vol. 67, Springer Science & Business Media, 2021.
- [33] W. E. Hart, J.-P. Watson, D. L. Woodruff, Pyomo: modeling and solving mathematical programs in python, *Mathematical Programming Computation* 3 (3) (2011) 219–260.
- [34] P. Kovács, Minimum-cost flow algorithms: an experimental evaluation, *Optimization Methods and Software* 30 (1) (2015) 94–127. doi:10.1080/10556788.2014.895828.

- [35] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, D. Weninger, Presolve reductions in mixed integer programming, *INFORMS Journal on Computing* 32 (2) (2020) 473–506. doi:10.1287/ijoc.2018.0857.
- [36] J. Castro, Solving difficult multicommodity problems with a specialized interior-point algorithm, *Annals of Operations Research* 124 (1) (2003) 35–48. doi:10.1023/B:ANOR.0000004761.99649.a5.