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

Felix P. Broesamle[a,*], Stefan Nickel[a]

[a]*Discrete Optimization and Logistics, Institute for Operations Research, Karlsruhe Institute of Technology, Kaiserstraße 89, Karlsruhe, 76133, Germany*

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 heterogeneity on computational methods, we propose two distinct integer partitioning schemes: *Uniform*, which is characterized by homogeneous distributions, and *Spread*, which introduces high degrees of heterogeneity. 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. Numerical experiments demonstrate that the SMCG is significantly influenced by commodity heterogeneity and serves as an indicator of the computational difficulty of multicommodity 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

*Corresponding author

Email addresses: felix.broesamle@kit.edu (Felix P. Broesamle), stefan.nickel@kit.edu (Stefan Nickel)

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 $\delta^{out}(v) \coloneqq \{(v, u) \in E \,|\, u \in V\}$ and $\delta^{in}(v) \coloneqq \{(u, v) \in E \,|\, u \in V\}$, respectively. The decision variable x_e represents the amount of flow assigned to arc e. The MCF problem is formulated as:

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

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

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

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 $K = \{1, \ldots, K\}$. 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:

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

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

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

$$0 \leq x_e^k \leq u_e^k, \qquad \forall e \in E, k \in K. \tag{2d}$$

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 and solvers such as `GCG` [5] and `Coluna` [6]. Comprehensive surveys on algorithms for the MCMCF are provided by [7, 8]. For specialized solution strategies, including interior point methods and decomposition methods, we refer to [9, 10, 11].

Recent advances include GPU-based interior point methods [12] and a refined polynomial reduction of general linear programs to maximum 2-commodity flow problems [13, 14].

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 [15, 16]. For single-commodity flows, the well-known `NETGEN` generator [17] has served as a primary source for decades, alongside specialized instances and established data formats, such as the DIMACS [18] `.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, or the widely utilized `MNETGEN` generator [19] and its refined variant [20]. A meta-generation approach based on stochastic scaling has been investigated in [21] and further improved in [20].

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. The contribution is threefold:

1. The meta-generation framework is built upon integer partitioning. Therefore, we propose two distinct integer partitioning strategies: *Uniform*, characterized by homogeneous commodity distributions, and *Spread*, which introduces high heterogeneity.
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.
3. To facilitate further research and reproducibility, the framework is implemented in `Rust` and released as the open-source `Python` library `s2mflow` [22].

The computational experiments in Section 4 reveal that solution methods exhibit different scaling behaviors depending on the chosen partitioning strategy. We find that the Barrier method requires significantly more computational effort for instances with high commodity heterogeneity (*Spread*), compared to homogeneous (*Uniform*) commodity demands. Conversely, the Dual Simplex method exhibits shorter solve times under high heterogeneity than in the homogeneous case. These findings demonstrate that the heterogeneity of commodity demands is not merely a parameter, but a critical structural characteristic for the systematic classification of MCMCF benchmarks 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 commodity heterogeneity 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 $b^k = B_{\cdot,k} \in \mathbb{Z}^n$ 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:

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

$$B_{vk} \in \mathbb{Z}, \qquad \forall v \in V, \forall k \in K, \tag{3b}$$

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

$$\sum_{v\in V} B_{vk} = 0, \qquad \forall k \in K. \tag{3d}$$

Constraint (3a) ensures role preservation: a supply node ($b_i > 0$) never becomes a demand node, and conversely, a demand node ($b_i < 0$) 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) and (3c) 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).

2.1. Phase 1: Local Partitioning Strategies

We propose two distinct regimes to control the degree of commodity heterogeneity, defined as the variance of commodity-specific supply and demand across the network.

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 $b_i \neq 0$, and $K \in \mathbb{N}$.
Output: Matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c).

Initialize $B \in \mathbb{Z}^{n \times K}$ with $B_{ik} \leftarrow \text{sign}(b_i)\left\lfloor \frac{|b_i|}{K} \right\rfloor$ for all $i = 1, \ldots, n, k = 1, \ldots, K$.
Let $next = 1$ ▷ Global remainder pointer.
for $i = 1 : n$ **do**
    $r \leftarrow |b_i| \mod K$.
    for $v = 1 : r$ **do**
        Update $B_{i,next} \leftarrow B_{i,next} + \text{sign}(b_i)$.
        Update $next \leftarrow (next \mod K) + 1$.
return B.

Theorem 2.1. *Given a vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$ for all $i = 1, \ldots, n$, 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 $k \in \{1, \ldots, K\}$.*

Proof. Algorithm 1 iterates through the n entries of b. For each $i \in \{1, \ldots, n\}$, the initialization of row $B_{i,\cdot}$ 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 $i = 1, \ldots, n$, let $|b_i| = q_i K + r_i$, where $q_i = \left\lfloor \frac{|b_i|}{K} \right\rfloor$ 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:

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

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. □

Example 2.2. *Consider the vector* $b = [7, 10, 4, -7, -10, -4]$*, which satisfies* $\sum_i b_i = 0$*. Applying Algorithm 1, we first perform the base assignment* $B_{ik} = \operatorname{sign}(b_i)\left\lfloor \frac{|b_i|}{K} \right\rfloor$*, 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.

Theorem 2.3. *Given a vector* $b \in \mathbb{Z}^n$*, where* $b_i \neq 0$ *for all* $i = 1, \ldots, n$*, 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* $\{0, \ldots, a\}$ *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

Algorithm 2 Integer Partitioning: *Spread*

Input: Vector $b \in \mathbb{Z}^n$, where $b_i \neq 0$, and $K \in \mathbb{N}$.
Output: Matrix $B \in \mathbb{Z}^{n \times K}$ satisfying (3a)–(3c).

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

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 $c_0, \ldots, c_K$. By a telescoping sum argument:

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

Since $c_k = |b_i|$ and $c_0 = 0$ by construction, we have $\sum_{k=1}^{K} B_{ik} = \operatorname{sign}(b_i)|b_i| = b_i$, satisfying (3c).
Sign Consistency: For each $k \in \{1, \ldots, K\}$, the value $(c_k - c_{k-1})$ is non-negative because the sequence $(c_j)_{j \in \{0,\ldots,K\}}$ is sorted. Consequently, $B_{ik} \geq 0$ if $b_i > 0$ and $B_{ik} \leq 0$ if $b_i < 0$, satisfying (3a).
Integrality: The matrix entries are defined as differences of integers, thus, B_{ik} is integer-valued. □

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

Example 2.4. *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* $\{0, \ldots, 13\}$*, 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* $\{0, \ldots, 2\}$*, 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 3.*

2.2. 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 3 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 2.5. *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 3 terminates and returns a matrix that satisfies* (3)*.*

Proof. We first show that Algorithm 3 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

Algorithm 3 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).

Initialize $\Delta_k \leftarrow \sum_{i=1}^{n} B_{ik}$ for all $k \in K$.
Initialize $p_{node} \leftarrow 1$ and $p_{def} \leftarrow 1$. ▷ Node and deficit commodity pointers.
while $\exists k : \Delta_k > 0$ **do**
    Let $\mathcal{K}^+ \leftarrow (k_1, ..., k_m)$ be a sequence of indices $\{k \mid \Delta_k > 0\}$.
    Let $\mathcal{K}^- \leftarrow (k_1, ..., k_q)$ be a sequence of indices $\{k \mid \Delta_k < 0\}$.
    if $\nexists k \in K$ with $\Delta_k > 0$ **then**
        break
    if $p_{def} > |\{k \mid \Delta_k < 0\}|$ **then**
        Set $p_{def} = 1$.
    Set $sk \leftarrow \mathcal{K}^+_1$. ▷ Pick surplus commodity to balance.
    for $step = 1 : n$ **do**
        Set $i \leftarrow p_{node}$.
        for $attempt = 1 : |\mathcal{K}^-|$ **do**
            Set $dk \leftarrow \mathcal{K}^-_{p_{def}}$.
            Set $can\_swap \leftarrow (b_i > 0 \wedge B_{i,sk} > 0) \vee (b_i < 0 \wedge B_{i,dk} < 0)$.
            Update $p_{def} \leftarrow (p_{def} \bmod |\mathcal{K}^-|) + 1$.
            if can_swap **then**
                Update $B_{i,sk} \leftarrow B_{i,sk} - 1$, and $B_{i,dk} \leftarrow B_{i,dk} + 1$.
                Update $\Delta_{sk} \leftarrow \Delta_{sk} - 1$, and $\Delta_{dk} \leftarrow \Delta_{dk} + 1$.
                break
        end for
        Update $p_{node} \leftarrow (p_{node} \bmod n) + 1$.
        if $\Delta_{sk} = 0$ **then**
            break
    end for
end while
return B

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 $\sum_{k=1}^{K} B_{ik} = b_i$ 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

($\Delta_{dk} < 0$).

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. □

We now continue Example 2.4 from the previous section to illustrate the balancing phase.

Example 2.6. *We continue from Example 2.4, 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: For $sk = 1, p_{node} = 1$, 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: For $sk = 3, p_{node} = 4$, 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 2.7. *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). 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 $\mathcal{P}_M = (V, E, K, u, \{u^k\}_{k \in K}, \{w^k\}_{k \in K}, \{b^k\}_{k \in K})$ be the corresponding multicommodity instance 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 $v \in V$:

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

Therefore, the single-commodity instance is feasible. □

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

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

and the relative Single-Multi-Commodity Gap rSMCG *as*

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

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

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 even if we relax the assumption that $u^k = u$ for all $k \in K$. We retain this assumption primarily to ensure that the SMCG continues to measure only the effect of demand partitioning rather than arbitrary capacity tightening.

The following theorem provides a sufficient condition for the feasibility of any MCMCF instance.

Theorem 3.4. *Let $\mathcal{P}_M = (V, E, K, u, \{u^k\}_{k\in K}, \{w^k\}_{k\in K}, \{b^k\}_{k\in K})$ be given. For each $k \in K$, let $D_k := \sum_{v\in V: b_v^k > 0} b_v^k$ denote the total supply of commodity k. The instance is guaranteed to be feasible if there exists a collection of arc-disjoint subgraphs $\{G_k = (V, E_k)\}_{k\in K}$ such that for each commodity $k \in K$:*

(1) $E_k \subseteq E$ and $E_k \cap E_j = \emptyset$ for all $k \neq j$.

(2) In G_k, every supply node $s \in S_k := \{v \in V \mid b_v^k > 0\}$ has a directed path to at least one demand node $t \in T_k := \{v \in V \mid b_v^k < 0\}$, and every demand node $t \in T_k$ is reachable from at least one supply node $s \in S_k$.

(3) For every arc $e \in E_k$, the capacities satisfy $u_e \geq D_k$ and $u_e^k \geq D_k$.

Proof. We prove feasibility by constructing a valid flow for each commodity k independently within its assigned subgraph G_k. By assumption (2), the connectivity of G_k is sufficient to send flow from the supply nodes S_k to the demand nodes T_k. Since every arc $e \in E_k$ has a capacity at least equal to the total supply D_k (3), any amount of flow required by the nodal demands can be sent through the network. This remains true even if the flow for commodity k is distributed over multiple paths that share a common arc, as the aggregate flow of a single commodity k on any arc e is inherently bounded by D_k.

Thus, there exists a flow vector $f^k \in \mathbb{R}_{\geq 0}^{|E|}$ satisfying (2c), with $f_e^k = 0$ for all $e \notin E_k$. By the arc-disjointness of the subgraphs (assumption 1), each arc $e \in E$ carries flow for at most one commodity. Therefore, for any $e \in E_k$, the bundle capacity constraint (2b) reduces to:

$$\sum_{j\in K} f_e^j = f_e^k \leq D_k \leq u_e.$$

The individual capacity constraints (2d) are satisfied similarly. The collection $\{f^k\}_{k\in K}$ thus constitutes a feasible solution for $\mathcal{P}_M$. □

4. Numerical Experiments

In this section, we evaluate the computational impact of transitioning from single-commodity to multicommodity flows using the meta-generation framework proposed

in Section 2. We specifically analyze the shift in computational tractability and the behavior of the (relative) Single-Multi-Commodity Gap (r)SMCG under varying degrees of commodity heterogeneity.

4.1. Experimental Setup and Methodology

All optimization models were implemented in `Python 3.13.5` using the `gurobipy 13.0.1` interface for the commercial solver `Gurobi 13.0.1` [23]. The computational experiments were conducted on a workstation equipped with 32 GB of RAM and an `Intel Core Ultra 7 255U` processor. The Gurobi parameter `Threads` was restricted to 1. We compare the Barrier Method (`Method` = 2) and the Dual Simplex method (`Method = 1`).

4.2. Base Single-Commodity Instances

The underlying networks were generated using the `netgen_generate` function from the `pynetgen 1.0.0` library [24], which provides a modern interface for the `NETGEN` generator [17]. We generated six base instances, I_1 through I_6. The fixed generation parameters for all instances are: `seed` = 12342001, `mincost` = 10, `maxcost` = 1000, `tsources`=0, `tsinks`=0, `hicost`=0, `capacitated=100`, `mincap`=50, `maxcap`=500, `rng` = 0. The instance-specific configurations are detailed in Table 1.

Table 1: Structural properties of the base single-commodity instances.

ID	Nodes (n)	Arcs (m)	Sources	Sinks	Total Supply
I_1	100	2000	8	8	4000
I_2	200	5000	16	16	8000
I_3	300	8000	24	24	16000
I_4	400	12000	32	32	24000
I_5	600	16000	48	48	32000
I_6	800	24000	64	64	40000

4.3. Multicommodity Instances

Using the `s2mflow` library [22] and the `generate_multi_commodity_data` method, each base instance $I \in \{I_1, \ldots, I_6\}$ was transformed into a suite of multicommodity instances with $K \in \{2, 5, 10, 25, 50\}$. To investigate the influence of commodity heterogeneity on solver performance and the SMCG, we generated two distinct variants for every (I, K) pair using both local partitioning methods from Section 2:

1. *Spread* (S): High heterogeneity generated via Algorithm 2 (`seed` = 12342001).
2. *Uniform* (U): Low heterogeneity generated via Algorithm 1.

4.4. Results

The problems were solved to optimality without a time limit. Tables 2 and 3 present the results for the *Spread* and *Uniform* partitioning strategies, respectively. We report the optimal objective values (ObjVal), the SMCG and rSMCG, and the runtimes for both the Barrier and Dual Simplex methods in seconds. A primary observation from Table 3 is that the *Uniform* partitioning method yields MCMCF instances with SMCG ≈ 0. This suggests that under homogeneous commodity distributions, the MCF obtained from aggregating the commodity demands acts as a strong relaxation of the MCMCF. In contrast, the *Spread* partitioning (Table 2) induces a significant objective difference, with rSMCG values ranging between 0.2% and 2.0%. This confirms that high heterogeneity in commodity distribution effectively "tightens" the network, forcing flows onto more expensive paths to satisfy nodal requirements.

We observe a significant increase in computation time as the number of commodities grows. Notably, the transition from a single-commodity instance to a two-commodity instance ($K = 2$) exhibits a disproportionately large increase in runtime.

The data reveals a significant performance difference between the solution methods. The Dual Simplex method consistently outperformed the Barrier method across all tested instances; for instance $(I_5, 25, S)$, Dual Simplex was more than 20 times faster.

Interestingly, the solver's performance scaled differently with respect to commodity heterogeneity: while the Barrier method required longer solve times for high-heterogeneity commodity demands (*Spread (S)*) than for homogeneous commodity demands (*Uniform (U)*), the Dual Simplex method showed shorter solve times for high heterogeneity commodity demands than for homogeneous commodity demands.

Altogether, these results demonstrate that the heterogeneity of commodity demands is a critical characteristic for the systematic classification of MCMCF benchmark instances. Consequently, heterogeneity of commodity demands should be considered a primary feature when evaluating the robustness of solution methods for the MCMCF.

5. Conclusions

In this work, we introduced a systematic meta-generation framework that bridges the gap between well-understood single-commodity and multicommodity instances. By "lifting" single-commodity models into the multicommodity space while strictly preserving the underlying network topology and nodal roles, we demonstrate that commodity demand heterogeneity is a fundamental determinant of both mathematical structure and computational tractability.

Table 2: Results for the *Spread* local partitioning.

ID	ObjVal	SMCG	rSMCG	Barrier Runtime	Dual Simplex Runtime
I_1	1 359 745	—	—	0.006	0.002
$(I_1, 2, S)$	1 363 567	3 822	0.00281	0.032	0.009
$(I_1, 5, S)$	1 378 749	19 004	0.01398	0.091	0.025
$(I_1, 10, S)$	1 373 043	13 298	0.00978	0.444	0.059
$(I_1, 25, S)$	1 376 469	16 724	0.01230	2.410	0.202
$(I_1, 50, S)$	1 372 727	12 982	0.00955	8.547	0.597
I_2	2 902 515	—	—	0.027	0.002
$(I_2, 2, S)$	2 949 637	47 122	0.01623	0.096	0.031
$(I_2, 5, S)$	2 950 168	47 653	0.01642	0.502	0.085
$(I_2, 10, S)$	2 933 245	30 730	0.01059	1.562	0.272
$(I_2, 25, S)$	2 946 127	43 612	0.01503	13.952	1.063
$(I_2, 50, S)$	2 945 295	42 780	0.01474	52.472	3.551
I_3	5 172 203	—	—	0.031	0.006
$(I_3, 2, S)$	5 228 215	56 012	0.01083	0.169	0.047
$(I_3, 5, S)$	5 226 740	54 537	0.01054	0.942	0.224
$(I_3, 10, S)$	5 238 060	65 857	0.01273	4.363	0.635
$(I_3, 25, S)$	5 250 082	77 879	0.01506	29.508	3.024
$(I_3, 50, S)$	5 236 135	63 932	0.01236	218.056	11.541
I_4	6 902 365	—	—	0.069	0.008
$(I_4, 2, S)$	6 952 677	50 312	0.00729	0.411	0.086
$(I_4, 5, S)$	6 999 817	97 452	0.01412	1.945	0.436
$(I_4, 10, S)$	6 954 911	52 546	0.00761	7.812	1.838
$(I_4, 25, S)$	6 937 159	34 794	0.00504	81.630	11.642
$(I_4, 50, S)$	6 939 501	37 136	0.00538	677.776	67.983
I_5	9 332 483	—	—	0.093	0.009
$(I_5, 2, S)$	9 405 326	72 843	0.00781	0.886	0.182
$(I_5, 5, S)$	9 445 781	113 298	0.01214	4.328	0.822
$(I_5, 10, S)$	9 517 566	185 083	0.01983	19.809	3.673
$(I_5, 25, S)$	9 463 455	130 972	0.01403	153.127	26.317
$(I_5, 50, S)$	9 420 958	88 475	0.00948	2947.905	124.316
I_6	10 932 747	—	—	0.169	0.021
$(I_6, 2, S)$	11 011 650	78 903	0.00722	1.375	0.199
$(I_6, 5, S)$	11 130 653	197 906	0.01810	6.639	1.324
$(I_6, 10, S)$	11 082 518	149 771	0.01370	31.429	7.318
$(I_6, 25, S)$	11 098 370	165 623	0.01515	340.752	62.102
$(I_6, 50, S)$	11 082 425	149 678	0.01369	3059.473	314.051

Table 3: Results for the *Uniform* local partitioning.

ID	ObjVal	SMCG	rSMCG	Barrier Runtime	Dual Simplex Runtime
I_1	1 359 745	—	—	0.006	0.002
$(I_1, 2, U)$	1 359 745	0	0	0.037	0.008
$(I_1, 5, U)$	1 359 745	0	0	0.096	0.027
$(I_1, 10, U)$	1 359 745	0	0	0.332	0.079
$(I_1, 25, U)$	1 359 745	0	0	0.497	0.134
$(I_1, 50, U)$	1 359 745	0	0	0.792	0.258
I_2	2 902 515	—	—	0.027	0.002
$(I_2, 2, U)$	2 902 515	0	0	0.087	0.021
$(I_2, 5, U)$	2 902 515	0	0	0.503	0.101
$(I_2, 10, U)$	2 902 515	0	0	1.399	0.400
$(I_2, 25, U)$	2 902 515	0	0	7.188	1.206
$(I_2, 50, U)$	2 902 515	0	0	13.487	1.811
I_3	5 172 203	—	—	0.031	0.006
$(I_3, 2, U)$	5 172 203	0	0	0.177	0.048
$(I_3, 5, U)$	5 172 203	0	0	0.886	0.220
$(I_3, 10, U)$	5 172 203	0	0	3.577	0.834
$(I_3, 25, U)$	5 172 203	0	0	20.498	3.629
$(I_3, 50, U)$	5 172 203	0	0	51.045	6.353
I_4	6 902 365	—	—	0.069	0.008
$(I_4, 2, U)$	6 902 365	0	0	0.470	0.092
$(I_4, 5, U)$	6 902 365	0	0	1.800	0.560
$(I_4, 10, U)$	6 902 365	0	0	6.351	2.292
$(I_4, 25, U)$	6 902 365	0	0	44.153	13.830
$(I_4, 50, U)$	6 902 952	587	0.00009	131.411	31.310
I_5	9 332 483	—	—	0.093	0.009
$(I_5, 2, U)$	9 332 483	0	0	0.827	0.170
$(I_5, 5, U)$	9 332 483	0	0	2.968	1.087
$(I_5, 10, U)$	9 332 483	0	0	13.417	4.241
$(I_5, 25, U)$	9 332 483	0	0	100.227	35.226
$(I_5, 50, U)$	9 332 483	0	0	602.518	106.820
I_6	10 932 747	—	—	0.169	0.021
$(I_6, 2, U)$	10 932 747	0	0	1.434	0.191
$(I_6, 5, U)$	10 932 747	0	0	6.753	2.003
$(I_6, 10, U)$	10 932 747	0	0	28.855	9.476
$(I_6, 25, U)$	10 932 747	0	0	270.604	95.281
$(I_6, 50, U)$	10 932 747	0	0	2621.584	430.234

The computational results reveal a distinct difference in the behavior of standard solution methods. While high commodity demand heterogeneity increases the computational effort required by the Barrier method, the Dual Simplex method conversely exhibits shorter solve times compared to instances with homogeneous commodity demands. Furthermore, the *(relative) Single-Multi-Commodity Gap* (r)SMCG provides a novel metric to quantify the divergence in optimal objective values arising from the transformation of a single-commodity instance to a multicommodity one.

These findings confirm that heterogeneity of commodity demand is not merely a parameter, but a critical structural characteristic for the systematic classification of MCMCF benchmarks and algorithmic performance. 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, Gemini 3) and DeepL Write 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, 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] 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.

[6] N. Javerzat, G. Marques, V. Nesello, A. Pessoa, R. Sadykov, F. Vanderbeck, Building coluna.jl, a branch-cut-and-price framework in julia (2023).

[7] A. A. Assad, Multicommodity network flows—a survey, Networks 8 (1) (1978) 37–91. doi:https://doi.org/10.1002/net.3230080107.

[8] 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

[9] 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.

[10] J. Castro, A specialized interior-point algorithm for multicommodity network flows, SIAM Journal on Optimization 10 (3) (2000) 852–877. doi:10.1137/S1052623498341879.

[11] 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.

[12] F. Zhang, S. Boyd, Solving large multicommodity network flow problems on gpus (2025). arXiv:2501.17996.
URL https://arxiv.org/abs/2501.17996

[13] A. Itai, Two-commodity flow, J. ACM 25 (1978) 596–611. doi:10.1145/322092.322100.

[14] 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.
URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2022.54

[15] A. Frangioni, Commalab: Datasets, single-commodity min-cost flow problems (2026).
URL https://commalab.di.unipi.it/datasets/mcf/

[16] A. Frangioni, Commalab: Datasets, multicommodity flow problems (2026).
URL https://commalab.di.unipi.it/datasets/mmcf/

[17] 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.

[18] DIMACS, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science (2026).
URL http://dimacs.rutgers.edu

[19] 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).

[20] 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.

[21] J. Castro, N. Nabona, Computational tests of a linear multicommodity network flow code with linear side constraints through primal partitioning, 2004.
URL https://api.semanticscholar.org/CorpusID:7023201

[22] F. P. Broesamle, S. Nickel, s2mflow: A meta-generator for multicommodity flow instances (2026).
URL https://github.com/FelixBroesamle/s2mflow

[23] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2026).
URL https://www.gurobi.com

[24] A. Rumpf, pynetgen 1.0.0 (2026).
URL https://pypi.org/project/pynetgen/