A LOCALIZED METHOD FOR MULTICOMMODITY FLOW PROBLEM

A PREPRINT

Pengfei Liu
Building No.1, Zhonguancun Road, Haidian District
Beijing, China
`pengfeil89@foxmail.com`

August 23, 2025

ABSTRACT

This paper introduces a novel theoretical framework and a suite of highly efficient, parallelizable algorithms for solving the large-scale multicommodity flow (MCF) feasibility problem. We reframe the classical constraint-satisfaction problem as an equilibrium search. By defining vertex-specific height functions and edge-specific congestion functions, we establish a new, intuitive optimality condition: a flow is feasible if and only if it corresponds to a zero-stable pseudo-flow, where all potential differences across the network are resolved. This condition gives rise to an edge-separable convex optimization problem, whose structure is inherently suited for massive parallelization.

Based on this formulation, we develop a family of Potential Difference Reduction (PDR) algorithms. Our primary method, provably convergent, solves an exact quadratic programming subproblem for each edge in parallel. To address scenarios with a very large number of commodities, we propose two computationally cheaper heuristics based on adaptive gradient descent. Extensive numerical experiments on well-known benchmarks demonstrate the framework's remarkable performance. This work provides a powerful new approach for tackling large-scale MCF problems, while also identifying the formal analysis of the convergence rate as a promising direction for future research.

***K*eywords** parallelizable algorithms · equilibrium search · potential difference reduction · multicommodity flow

1 Introduction

The *multicommodity flow problem* involves designing the flow for several different commodities through a common network with varying edge capacities. We are given a directed graph $G(V,E)$ with n vertices and m edges, a capacity function $u : E \to \mathbb{Q}^+$, and K commodities. Each commodity k is defined by an origin-destination pair (s_k, t_k) and a demand d_k. The objective is to find a flow assignment that satisfies the demand for each commodity without violating edge capacity constraints. The constraints can be summarized as follows:

$$
\begin{aligned}
&\sum_{k\in\mathcal{K}} f_{ij,k} \le u_{ij}, \forall (i,j) \in E\\
&f_{ij,k} \ge 0, \forall k \in \mathcal{K}, (i,j) \in E,\\
&\sum_{j\in\delta^+(i)} f_{ij,k} - \sum_{j\in\delta^-(i)} f_{ji,k} = \begin{cases} d_k, \text{ if } i = s_k\\ -d_k, \text{ if } i = t_k\\ 0, \text{ if } i \in V - \{s_k, t_k\}\end{cases}\\
&\forall k \in \mathcal{K}, i \in V
\end{aligned}
\tag{1}
$$

where $\delta^+(i) = \{j|(i,j) \in E\}, \delta^-(i) = \{j|(j,i) \in E\}$ and $\mathcal{K} = \{1, 2, \cdots, K\}$. In this paper, we assume $s_k \neq t_k$. The first expressions are capacity constraints. The second are non-negative constraints, and the last are flow conservation constraints.

Conventional centralized MCF algorithms require a central entity with complete network knowledge to compute optimal flows. In large-scale distributed networks, gathering this global information incurs significant communication overhead and delays (Farrugia et al., 2023). Moreover, centralized approaches lack robustness due to a single point of failure and adapt poorly to dynamic network conditions. They are also unsuitable for scenarios where vertices must make autonomous decisions based on local information, as is common in ad-hoc networks or systems with dynamic membership (Awerbuch et al., 2007).

With the advancement of AI, the architectural paradigm of modern high-performance computing has shifted decisively towards massive parallelism (Robey and Zamora, 2021; D'Agostino et al., 2021). Classical optimization algorithms designed for single-core processors often fail to exploit this potential. This technological shift creates a compelling need for algorithms whose computational structure mirrors the distributed nature of the network itself, making them both localized and inherently parallelizable.

This paper introduces a novel algorithmic framework for the multicommodity flow problem that directly addresses the limitations of existing methods by eliminating any reliance on global information. Departing from traditional methods, our approach simultaneously relaxes both capacity and flow conservation constraints. This complete relaxation transforms the problem into an equilibrium-finding problem governed by local interactions, guided by the physical analogy of flow moving from high to low potential.

To formalize this intuition, we introduce a new theoretical framework built upon local functions for congestion, nodal imbalance (height), and a "driving force" we term potential difference. This leads to the definition of a stable pseudo-flow. Our central theoretical result is that a feasible solution to the MCF problem exists if and only if a zero-stable pseudo-flow exists, which provides a complete and locally verifiable optimality condition. From this, we designed the *Potential Difference Reduction* (PDR) Algorithm, a simple iterative process that adjusts flows based on local rule, making the process inherently parallelizable.

The network notation introduced here is summarized in Table 1. Further notation is introduced as needed.

Table 1: Basic Network Notation

$G(V,E)$	a directed graph
V	vertex (index) set
E	edge (index) set
$\mathcal{K}$	set of commodities, i.e., $\mathcal{K} = \{1, \cdots, k, \cdots, K\}$
$f_{ij,k}$	flow of commodity k on edge (i,j), $\boldsymbol{f} = (\cdots, f_{ij,k}, \cdots)$
f_{ij}	flow on edge (i,j), i.e., $f_{ij} = \sum_{k\in\mathcal{K}} f_{ij,k}$
(s_k, t_k, d_k)	s_k and t_k are the origin and destination of commodity k, and d_k is the demand
u_{ij}	the capacity of edge (i,j)
$\delta^+(i)$	$\{j\|(i,j) \in E\}$
$\delta^-(i)$	$\{j\|(j,i) \in E\}$
δ_i	the degree of vertex i
ψ_{ij}	congestion function of edge (i,j)
h_{ik}	height function of vertex i for commodity k
Δ_{ik}	the amount by which the kth flow into the vertex i exceeds that out of the vertex
r_{ij}	the unused capacity of edge (i,j)
m	the number of edges of graph $G(V,E)$
n	the number of vertices of graph $G(V,E)$
$\phi_{ij,k}$	the potential difference across edge (i,j) for commodity k

1.1 Our Contribution

This paper presents a novel theoretical framework and a suite of highly efficient, parallelizable algorithms for solving the multicommodity flow feasibility problem. Our contributions are multi-faceted and can be summarized as follows:

1. **A Novel Optimality Condition:** By defining **vertex-specific height functions** and **edge-specific congestion functions**, we establish a simple and intuitive optimality condition, which reframes the MCF problem from satisfying a set of hard constraints to finding an equilibrium state.

2. **An Edge-Separable Convex Formulation:** We formulate the stable pseudo-flow search as a convex optimization problem, yielding an inherent edge-separable structure, which makes the approach ideal for parallel processing on modern GPUs.
3. **A Convergent, Decomposed Algorithm:** Based on the edge-separable formulation, we propose the *Potential Difference Reduction Algorithm via Exact Subproblem Optimization*, which iteratively solves a small quadratic program for each edge in parallel.
4. **Scalable Heuristics and Empirical Insights:** Two computationally cheaper heuristic algorithms are developed based on adaptive gradient descent, and an enhanced variant, which incorporates a **momentum term**, is specifically designed to accelerate convergence.
5. **An Open Question for Theoretical Analysis:** While we have proven the convergence of the proposed methods, a formal analysis of their convergence rate remains an open problem. We posit that establishing a *tight theoretical bound* on the rate is a non-trivial challenge, likely dependent on key topological and spectral properties of the underlying network graph.

2 Related Work

Multicommodity flow (MCF) problems have attracted significant attention since the seminal works of Ford and Fulkerson (2015) and Hu (1963). For a comprehensive survey of the problem, see Wang (2018a,b); Salimifard and Bigharaz (2022).

Many specialized solution methods based on linear programming have been proposed to exploit the block-angular structure inherent in MCF problems (Kennington, 1977; Geoffrion, 1970; Shetty and Muthukrishnan, 1990; Frangioni and Gallo, 1999; Goffin et al., 1997; Moura et al., 2018; Zenios et al., 1995; Kamath and Palmon, 1995; Kapoor and Vaidya, 1996; Cohen et al., 2021). Conversely, studies by Itai (1978) and Ding et al. (2022) have demonstrated that general linear programs can be transformed into two-commodity flow problems through nearly-linear time reductions.

Approximation algorithms for the MCF problem are also numerous (Leighton and Rao, 1999; Brun et al., 2017; Garg and Könemann, 2007; Leighton et al., 1995; Schneur and Orlin, 1998; Shahrokhi and Matula, 1990; Babonneau et al., 2006; Chen and Ye, 2023). Awerbuch and Leighton (1993, 1994) proposed an approximation algorithm that uses local-control techniques similar to the preflow-push algorithm of Goldberg and Tarjan (1987, 1988). Unlike previous methods that find shortest or augmenting paths to push flow, their algorithm uses an "edge-balancing" technique that sends a commodity across an edge (i, j) if the commodity is more queued at vertex i than at vertex j. The push-relabel algorithm by Goldberg and Tarjan (1987, 1988), which relaxes flow conservation constraints, is considered one of the most efficient algorithms for the maximum-flow problem. Recent improvements in this area compute the max-flow via a sequence of electric flows (Spielman and Teng, 2004; Daitch and Spielman, 2008; Christiano et al., 2011; Kelner et al., 2014; Madry, 2016; Axiotis et al., 2022; Gao et al., 2022; Chen et al., 2022; Van Den Brand et al., 2023). Building on this, Brand and Zhang (2023) developed a high-accuracy algorithm for the multicommodity flow problem using single-commodity methods.

Other approaches include using network equilibrium models to solve MCF problems (Liu, 2019; Bùi, 2022; Yu et al., 2022) and pseudo-flow methods for the max-flow problem (Hochbaum, 2008; Chandran and Hochbaum, 2009; Fishbain et al., 2010). Flow decomposition has also been explored, often leveraging the problem's decomposable nature by commodity (Hartman et al., 2012; Haeupler et al., 2024).

Furthermore, research has explored parallel and distributed approaches for solving the MCF problem (Awerbuch and Leighton, 1993, 1994; Zhang and Boyd, 2025; Castro and Frangioni, 2000; Farrugia et al., 2023; Ghaffari et al., 2015; Awerbuch et al., 2007). However, the effectiveness of parallelization can be limited by synchronization overhead and the inherently sequential components of some algorithms.

3 Optimality Condition

3.1 Stable pseudo-flow

Unlike other methods, our method *does not enforce the capacity and flow conservation constraints*. The method, however, maintains a *pseudo-flow*, which is a function $\boldsymbol{f} : K \times E \to \mathbb{R}^+$. Let Δ_{ik} be the amount by which the flow of commodity k into the vertex i exceeds the flow out of it, i.e.,

$$\Delta_{ik} = \sum_{j\in\delta^-(i)} f_{ji,k} - \sum_{j\in\delta^+(i)} f_{ij,k} + \hat{\Delta}_{ik}, \forall i \in V, k \in \mathcal{K}, \tag{2}$$

where $\delta^+(i) = \{j|(i,j) \in E\}$, $\delta^-(i) = \{j|(j,i) \in E\}$ and $\hat{\Delta}_{ik}$ is defined as:

$$\hat{\Delta}_{ik} = \begin{cases} 0 & if\ i \in V - \{s_k, t_k\} \\ d_k & if\ i = s_k \\ -d_k & if\ i = t_k. \end{cases} \tag{3}$$

For each commodity k at each vertex i , a height function $h_{ik} = h_{ik}(\Delta_{ik})$ is introduced, where $h_{ik}(\cdot)$ represents the relationship between Δ_{ik} and the height of vertex i for commodity k. Specifically, h_{ik} is defined as

$$h_{ik}(\Delta_{ik}) = \frac{\Delta_{ik}}{\delta_i}, \tag{4}$$

where δ_i is the degree of vertex i.

That is, the height of vertex i for commodity k is defined as the excess of inflow over outflow of commodity k at that vertex, normalized by the degree of vertex i.

Additionally, a *congestion function* $\psi_{ij} = \psi_{ij}(f_{ij})$ for each edge is introduced, where $\psi_{ij}(\cdot)$ represents the relationship between the flow and the degree of congestion for edge (i,j). Specifically, ψ_{ij} is defined as

$$\psi_{ij}(f_{ij}) = \begin{cases} 0 & if\ f_{ij} \le u_{ij} \\ f_{ij} - u_{ij} & if\ f_{ij} > u_{ij}, \end{cases} \tag{5}$$

where u_{ij} is the capacity of edge (i,j) and $f_{ij} = \sum_{k \in \mathcal{K}} f_{ij,k}$. If the flow on edge (i,j) is less than the capacity, the congestion of edge (i,j) is zero. Otherwise, the congestion of edge (i,j) is the amount by which the flow exceeds the capacity.

The greater the h_{ik} is, the higher the vertex i for commodity k is. The greater the ψ_{ij} is, the more congested the edge (i,j) is. As is often said, water finds its level. Intuitively, if the height difference between vertex i and vertex j for commodity k is greater than the congestion of edge (i,j), the flow $f_{ij,k}$ should be increased; if the height difference is less than the congestion, the flow $f_{ij,k}$ should be decreased. In the remainder of this paper, , we provide a rigorous proof to demonstrate that the feasible solution for the multicommodity flow problem can be obtained by this intuitive idea.

First, we introduce the concepts of *potential difference* and *stable pseudo-flow*.

Definition 1 *For every edge (i,j) and commodity k, the* potential difference *$\phi_{ij,k}$ is defined as the height difference between vertex i and vertex j, minus the congestion of edge (i,j), i.e.,*

$$\phi_{ij,k} = h_{ik} - h_{jk} - \psi_{ij}. \tag{6}$$

Definition 2 *By using { $\psi_{ij}, \forall (i,j) \in E$} as the congestion function and { $h_{ik}, \forall i \in V, \forall k \in \mathcal{K}$} as the height function, a pseudo-flow $\boldsymbol{f}$ is called a* stable pseudo-flow *if it satisfies the following conditions:*

(i) for any used edge of commodity k, the height difference between vertex i and vertex j for commodity k is equal to the congestion of edge (i,j), i.e., the potential difference $\phi_{ij,k}$ is zero;

(ii) for any unused edge of commodity k, the height difference between vertex i and vertex j is less than or equal to the congestion of edge (i,j), i.e., the potential difference $\phi_{ij,k}$ is less than or equal to zero;

where an edge (i,j) is called *used* by commodity k if there exists $s_k - t_k$ flow on edge (i,j), otherwise it is called *unused*.

Definition 3 *A stable pseudo-flow $\boldsymbol{f}$ is called* zero-stable *pseudo-flow if $\psi_{ij} = 0$ and $h_{ik} = 0$, for all $(i,j) \in E, i \in V, k \in \mathcal{K}$. Otherwise, it is called* nonzero-stable *pseudo-flow.*

From the definitions above, a zero-stable pseudo-flow is a feasible flow that satisfies Expression (1). Therefore, we have the following theorem:

Theorem 1 *Given $\{(s_k, t_k, d_k) : k \in \mathcal{K}\}$ and edge capacities $\{u_{ij} : (i,j) \in E\}$, the feasible region of Expression (1) is not empty if and only if there exists a zero-stable pseudo-flow.*

In fact, if a nonzero-stable pseudo-flow exists, there is no feasible solution for Expression (1).

3.2 Basic Formulation

Let f_{ij} be the sum of the flow of all commodities on edge (i, j) and Δ_{ik} defined as Expression (2). Define the following programming:

$$\begin{aligned} \textbf{min} \quad & z = \sum_{(i,j)\in E} \int_0^{f_{ij}} \psi_{ij}(\omega) d\omega \\ & + \sum_{i,k} \int_0^{\Delta_{ik}} h_{ik}(\omega) d\omega \\ \textbf{s.t} \quad & f_{ij,k} \geq 0, \forall k \in \mathcal{K}, (i,j) \in E \end{aligned} \tag{7}$$

where ψ_{ij} is the congestion function and h_{ik} the height function.

In the above programming, the objective function is the sum of the integrals of the edge congestion functions and the integrals of the vertex height functions. It should be noted that there are only non-negative constraints present and no capacity constraint or flow conservation constraint. According to the definitions of the congestion function denoted by ψ and the height function denoted by h, the following lemma can be derived:

Lemma 1 *The feasible region of Expression (1) is not empty if and only if the minimum value of the objective function of Expression (7) is zero. Conversely, the feasible region of Expression (1) is empty if and only if the minimum value of the objective function of Expression (7) is greater than zero.*

3.3 Equivalence

To demonstrate the equivalence between the stable pseudo-flow and the optimal solution of Programming (7), it has to be shown that any flow pattern that solves Programming (7) satisfies the stable conditions. This equivalence is demonstrated by proving that the Karush-Kuhn-Tucker conditions for Programming (7) are identical to the stable conditions.

Lemma 2 *Let $\boldsymbol{f}^*$ be a solution of Programming (7). $\boldsymbol{f}^*$ is the optimal solution of Programming (7) if and only if it satisfies the Karush-Kuhn-Tucker conditions of Programming (7).*

Proof: Firstly, the objective function of Programming (7) is convex. Secondly, the inequality constraints of Programming (7) are continuously differentiable concave functions. Therefore, Karush-Kuhn-Tucker conditions are necessary and sufficient for the optimality of Programming (7) (see Boyd et al. (2004)).

□

Since Programming (7) is a minimization problem with non-negativity constraints, the Karush-Kuhn-Tucker conditions of such formulation are as follows:

$$\begin{aligned} & \textbf{Stationarity} \\ & \quad -\frac{\partial z}{\partial f_{ij,k}} = -\mu_{ij,k}, \forall k \in \mathcal{K}, (i,j) \in E \\ & \textbf{Primal feasibility} \\ & \quad -f_{ij,k} \leq 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & \textbf{Dual feasibility} \\ & \quad \mu_{ij,k} \geq 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & \textbf{Complementary slackness} \\ & \quad \mu_{ij,k} f_{ij,k} = 0, \forall k \in \mathcal{K}, (i,j) \in E. \end{aligned} \tag{8}$$

Obviously,

$$\begin{aligned} \frac{\partial z}{\partial f_{ij,k}} &= \psi_{ij}(f_{ij}) \frac{\partial f_{ij}}{\partial f_{ij,k}} \\ &+ h_{ik}(\Delta_{ik}) \frac{\partial \Delta_{ik}}{\partial f_{ij,k}} + h_{jk}(\Delta_{jk}) \frac{\partial \Delta_{jk}}{\partial f_{ij,k}} \\ &= \psi_{ij}(f_{ij}) + (-h_{ik}(\Delta_{ik})) + (h_{jk}(\Delta_{jk})) \\ &= \psi_{ij} + h_{jk} - h_{ik}. \end{aligned} \tag{9}$$

Substituting the expression above into the Stationarity expression in KKT conditions,

$$h_{ik} - h_{jk} - \psi_{ij} = -\mu_{ij,k}, \forall k \in \mathcal{K}, (i,j) \in E.$$

For any used edge of commodity k, i.e. $f_{ij,k} > 0$, by complementary slackness $\mu_{ij,k} f_{ij,k} = 0$ in KKT conditions, we have $\mu_{ij,k} = 0$. Therefore,

$$h_{ik} - h_{jk} - \psi_{ij} = 0, \forall k \in \mathcal{K}, (i,j) \in E.$$

That is, the potential difference between vertex i and vertex j for commodity k is zero for any used edge of commodity k.

For any unused edge of commodity k, due to $\mu_{ij,k} \geq 0$, we have

$$h_{ik} - h_{jk} - \psi_{ij} = -\mu_{ij,k} \leq 0, \forall k \in \mathcal{K}, (i,j) \in E.$$

This direct interpretation of the Karush-Kuhn-Tucker conditions allows us to formulate the optimality conditions in terms of the potential difference $\phi_{ij,k}$, which are equivalent to the complementary slackness conditions for this problem.

Definition 4 (Stable Flow Optimality Conditions) *A pseudo-flow $\boldsymbol{f}$ satisfies the optimality conditions for Programming (7) if and only if for every commodity $k \in \mathcal{K}$ and every edge $(i,j) \in E$:*

(i) if the edge is used ($f_{ij,k} > 0$), the potential difference is zero:

$$\phi_{ij,k} = h_{ik} - h_{jk} - \psi_{ij} = 0;$$

(ii) if the edge is unused ($f_{ij,k} = 0$), the potential difference is non-positive:

$$\phi_{ij,k} = h_{ik} - h_{jk} - \psi_{ij} \leq 0.$$

These conditions are precisely the definition of a stable pseudo-flow (see Definition 2). This equivalence immediately leads to the following lemma:

Lemma 3 *The optimal solution of Programming (7) is a stable pseudo-flow.*

Obviously, a stable pseudo-flow also satisfies the KKT conditions. By Lemma 2, we have

Lemma 4 *A stable pseudo-flow is the optimal solution of Programming (7).*

By Lemma 1, Lemma 3 and Lemma 4, Theorem 2 holds.

Theorem 2 *Given $\{(s_k, t_k, d_k) : k \in \mathcal{K}\}$ and edge capacities $\{u_{ij} : (i,j) \in E\}$, the feasible region of Expression (1) is empty if and only if there exists nonzero-stable pseudo-flow.*

By Theorem 1 and Theorem 2, the *optimality condition* for multicommodity flow problem may be summarized as follows:

(i) if there exists a nonzero-stable pseudo-flow, there exists no feasible solution for the multicommodity flow problem;

(ii) if there exists a zero-stable pseudo-flow, there exists a feasible solution and the zero-stable pseudo-flow is the feasible solution.

Remark 1 *In fact, both Theorem 1 and Theorem 2 are true if the height function and the congestion function satisfy the following conditions:*

(i) the height function $h_{ik}(\Delta_{ik})$ is a strictly monotone increasing function and $h_{ik}(0) = 0$;

(ii) the congestion function ψ_{ij} satisfies the following definition:

$$\psi(f_{ij}) = \begin{cases} 0 & if\ f_{ij} \leq u_{ij} \\ g(f_{ij} - u_{ij}) & if\ f_{ij} > u_{ij}, \end{cases}$$

where $g(0) = 0$ and $g(\cdot)$ is a strictly monotone increasing function.

Remark 2 *Expression (9) shows that the potential difference is exactly the same as the negative gradient of the objective function of Programming (7).*

3.4 Separable Structure

The objective function of Programming (7) has a mixed structure: the first term involves a sum over edges (i, j), while the second term involves a sum over vertices i. The flow variables $f_{ij,k}$ are coupled in a complex way across both edges and vertices. We now rewrite its objective function as a single, unified sum over edges (i, j).

Firstly,

$$\begin{aligned}\int_0^{\Delta_{ik}} h_{ik}(\omega)d\omega &= \frac{1}{2\delta_i}\Delta_{ik}^2 \\ &= \frac{1}{2\delta_i}(\sum_{j\in\delta^-(i)} f_{ji,k} - \sum_{j\in\delta^+(i)} f_{ij,k} + \hat{\Delta}_{ik})^2.\end{aligned} \tag{10}$$

By the definition of congestion function ψ_{ij}, we have

$$\int_0^{f_{ij}} \psi_{ij}(\omega)d\omega = \begin{cases} 0 & if\ f_{ij} \le u_{ij} \\ \frac{1}{2}(f_{ij} - u_{ij})^2 & if\ f_{ij} > u_{ij}. \end{cases} \tag{11}$$

By introducing an auxiliary variable $\{r_{ij} : r_{ij} \ge 0\}$ which is the unused capacity for edge (i, j), Programming (7) could be rewritten as

$$\begin{aligned} \mathbf{min}\quad & z = \frac{1}{2}\sum_{(i,j)\in E}(\sum_{k\in\mathcal{K}} f_{ij,k} + r_{ij} - u_{ij})^2 \\ & + \sum_{i,k}(\frac{1}{2\delta_i}(\sum_{j\in\delta^-(i)} f_{ji,k} - \sum_{j\in\delta^+(i)} f_{ij,k} + \hat{\Delta}_{ik})^2) \\ & = \frac{1}{2}\sum_{(i,j)\in E}[(\sum_{k\in\mathcal{K}} f_{ij,k} + r_{ij} - u_{ij})^2 \\ & + \sum_{k\in\mathcal{K}}(\frac{\sum_{q\in\delta^-(i)} f_{qi,k} - \sum_{q\in\delta^+(i)} f_{iq,k} + \hat{\Delta}_{ik}}{\delta_i})^2 \\ & + \sum_{k\in\mathcal{K}}(\frac{\sum_{q\in\delta^-(j)} f_{qj,k} - \sum_{q\in\delta^+(j)} f_{jq,k} + \hat{\Delta}_{jk}}{\delta_j})^2] \\ & = \frac{1}{2}\sum_{(i,j)\in E}[\psi_{ij}^2 + \sum_{k\in\mathcal{K}} h_{ik}^2 + \sum_{k\in\mathcal{K}} h_{jk}^2] \\ \mathbf{s.t}\quad & f_{ij,k} \ge 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & r_{ij} \ge 0, \forall (i,j) \in E \end{aligned} \tag{12}$$

The equivalence between the vertex-based and edge-based summations is established by a simple combinatorial principle. When a term is summed over all edges, any property related to the endpoints of those edges (like the height function h_{ik}) is counted for each vertex precisely as many times as its degree. In the expression $\sum_{(i,j)\in E}[h_{ik}^2 + h_{jk}^2]$, a specific vertex v appears in the sum once for each edge connected to it. Therefore, the term h_{vk}^2 is counted δ_v times, justifying the transformation.

The objective function's edge-based terms are ideal for *decomposition algorithms*, which split the problem into smaller, independent *subproblems* per edge. These can be solved *in parallel*, *significantly* accelerating solutions for large networks.

4 PDR via Exact Subproblem Optimization

Theorem 1 and Theorem 2 above suggest a simple algorithmic approach for solving the multicommodity flow problem. That is, design an algorithm to obtain the stable pseudo-flow, which may be achieved by adjusting the commodity flow $f_{ij,k}$ until the potential difference $\phi_{ij,k} = 0$ or $f_{ij,k} = 0$. In this paper we call the algorithm the *potential difference reduction* (PDR) algorithms.

The preceding section established that the multicommodity flow problem can be formulated as the convex optimization problem in Expression (12), which seeks a zero-stable pseudo-flow. The parallel algorithm proposed here is to iteratively adjust the flows on each edge to locally minimize the objective function.

4.1 The Edge Subproblem

For each edge $(i, j) \in E$, we define an edge subproblem, which is derived from the terms in the global objective function (Expression 12). The objective is to minimize a function representing the new congestion on edge (i, j) plus the change in "height energy" at vertices i and j caused by the flow update on that single edge. This yields the following Quadratic Program (QP):

$$\begin{aligned}
\textbf{min} \quad & z_{ij} = \frac{1}{2}[(\sum_{k\in\mathcal{K}} f_{ij,k} + \sum_{k\in\mathcal{K}} \Delta f_{ij,k} + r_{ij} - u_{ij})^2 \\
& + \sum_{k\in\mathcal{K}} (h_{ik} - \Delta f_{ij,k})^2 + \sum_{k\in\mathcal{K}} (h_{jk} + \Delta f_{ij,k})^2] \\
\textbf{s.t} \quad & \Delta f_{ij,k} \geq -f_{ij,k}, \forall k \in \mathcal{K} \\
& r_{ij} \geq 0
\end{aligned} \tag{13}$$

These terms, $(h_{ik} - \Delta f_{ij,k})$ and $(h_{jk} + \Delta f_{ij,k})$, reflect the local impact of the flow adjustment on node heights: as a flow increment $\Delta f_{ij,k}$ moves from node i to node j, it effectively lowers the height at i while raising the height at j.

Each edge subproblem is solved in parallel, based on the height information from the previous iteration. Note that it does not account for the simultaneous flow changes happening on other edges connected to vertices i and j.

Remark 3 *Since this subproblem is defined for a* ***single edge****, its local contribution to the height update is treated as if the degree were 1. This explains why the flow change $\Delta f_{ij,k}$ is subtracted directly from the height h_{ik}, rather than being scaled by the total vertex degree δ_i.*

4.2 Algorithm Description

The proposed algorithm, which we term **PDR via Exact Subproblem Optimization**, iteratively solves these edge subproblems until the system converges to a stable pseudo-flow. The steps are outlined in Algorithm 1.

Each iteration begins with parallel computations of the flow imbalance Δ_{ik}, vertex heights h_{ik}, and edge congestions ψ_{ij}. The algorithm then checks for convergence by determining if the change in the objective function value between iterations is within a specified tolerance ϵ_{obj}. If not, it proceeds to the main update step, where it solves the QP edge subproblem in Expression (14) for each edge (i, j) to find the flows for the next iteration, $f_{ij,k}^{(t+1)}$. Note that Expression (14) is simply a variable substitution of Expression (13).

Crucially, since each subproblem depends only on local information, these optimizations can be executed **in parallel** for all edges. This parallel structure makes the algorithm exceptionally scalable and well-suited for large networks.

$$\begin{aligned}
\textbf{min} \quad & z_{ij} = \frac{1}{2}[(\sum_{k\in\mathcal{K}} f_{ij,k}^{(t+1)} + r_{ij} - u_{ij})^2 \\
& + \sum_{k\in\mathcal{K}} (h_{ik}^{(t)} + f_{ij,k}^{(t)} - f_{ij,k}^{(t+1)})^2 \\
& + \sum_{k\in\mathcal{K}} (h_{jk}^{(t)} - f_{ij,k}^{(t)} + f_{ij,k}^{(t+1)})^2] \\
\textbf{s.t} \quad & f_{ij,k}^{(t+1)} \geq 0, \forall k \in \mathcal{K} \\
& r_{ij} \geq 0
\end{aligned} \tag{14}$$

Algorithm 1 PDR via Exact Subproblem Optimization for MCF

```
1: Input: Graph $G(V,E)$, capacities $\{u_{ij}\}$, commodities $\{(s_k,t_k,d_k)\}$, tolerance $\epsilon_{\text{obj}} > 0$.
2: Initialize: Set initial flow $f^{(0)} \leftarrow 0$, iteration $t \leftarrow 0$, previous objective value $z_{\text{prev}} \leftarrow \infty$.
3: while $t <$ max_iterations do
4:     for all vertices $i \in V$, commodities $k \in \mathcal{K}$ in parallel do
5:         Flow imbalance $\Delta_{ik}^{(t)} \leftarrow \sum_{j\in\delta^-(i)} f_{ji,k}^{(t)} - \sum_{j\in\delta^+(i)} f_{ij,k}^{(t)} + \hat{\Delta}_{ik}$
6:         Vertex height $h_{ik}^{(t)} \leftarrow \Delta_{ik}^{(t)}/\delta_i$
7:     end for
8:     for all edges $(i,j) \in E$ in parallel do
9:         Edge congestion $\psi_{ij}^{(t)} \leftarrow \max\left(0, \sum_{k\in\mathcal{K}} f_{ij,k}^{(t)} - u_{ij}\right)$
10:    end for
11:                                                   ▷ Compute current objective value $z^{(t)}$
12:    $z^{(t)} \leftarrow \frac{1}{2}\sum_{(i,j)\in E}(\psi_{ij}^{(t)})^2 + \frac{1}{2}\sum_{i\in V,k\in\mathcal{K}} \delta_i (h_{ik}^{(t)})^2$
13:                              ▷ Check for convergence based on objective value change
14:    if $|z^{(t)} - z_{\text{prev}}| < \epsilon_{\text{obj}}$ then
15:        break
16:    end if
17:    $z_{\text{prev}} \leftarrow z^{(t)}$                ▷ Update previous objective value for the next iteration
18:                                             ▷ Update flows by solving edge subproblems
19:    For each edge $(i,j) \in E$ in parallel, solve the QP subproblem in Expression (14) to obtain the updated flows
       $f_{ij,k}^{(t+1)}$ for all $k \in \mathcal{K}$.
20:    $t \leftarrow t+1$
21: end while
22: Return final flow $f^{(t)}$
```

4.3 Proof of Algorithm Convergence

In this section, we prove that Algorithm 1, is convergent. The core idea is to show that the value of the global objective function z, defined in Expression (12), is monotonically non-increasing in each iteration. Since z is a sum of squared terms, it is clearly bounded below ($z \geq 0$). A monotonically non-increasing sequence that is bounded below is guaranteed to converge.

Theorem 3 *For the sequence of flows $\{\boldsymbol{f}^{(t)}\}_{t=0,1,2,\cdots}$ generated by Algorithm 1, the corresponding sequence of global objective function values $\{z(\boldsymbol{f}^{(t)})\}$ is monotonically non-increasing. That is:*

$$z(\boldsymbol{f}^{(t+1)}) \leq z(\boldsymbol{f}^{(t)}), \quad \forall t \geq 0$$

Proof: To prove this theorem, we introduce a surrogate function and complete the proof in two steps.

Let $\boldsymbol{f}^{(t)}$ be the flow vector at the start of iteration t. The algorithm computes the new flow vector $\boldsymbol{f}^{(t+1)}$ by solving, in parallel, the QP subproblems (Expression (14)) for all edges $(i,j) \in E$. The sum of the objective functions of all these subproblems can be formulated as a surrogate function $Q(\boldsymbol{f}', \boldsymbol{f}^{(t)})$, where $\boldsymbol{f}'$ is the variable to be optimized. Based on Expression (14), we define Q as:

$$Q(\boldsymbol{f}', \boldsymbol{f}^{(t)}) = \frac{1}{2}\sum_{(i,j)\in E}\left[\left(\sum_{k\in\mathcal{K}} f'_{ij,k} + r'_{ij} - u_{ij}\right)^2 + \sum_{k\in\mathcal{K}}\left(h_{ik}^{(t)} - (f'_{ij,k} - f_{ij,k}^{(t)})\right)^2 + \sum_{k\in\mathcal{K}}\left(h_{jk}^{(t)} + (f'_{ij,k} - f_{ij,k}^{(t)})\right)^2\right]$$

Here, $h_{ik}^{(t)}$ and $h_{jk}^{(t)}$ are the vertex heights calculated based on the flow $\boldsymbol{f}^{(t)}$.

Step 1: Proving that $Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)}) \leq z(\boldsymbol{f}^{(t)})$

By the definition of Algorithm 1, $\boldsymbol{f}^{(t+1)}$ is obtained by minimizing $Q(\boldsymbol{f}', \boldsymbol{f}^{(t)})$, as this minimization is composed of the independent minimizations of each edge subproblem. Therefore, $\boldsymbol{f}^{(t+1)}$ is the global minimizer of $Q(\boldsymbol{f}', \boldsymbol{f}^{(t)})$.

$$\boldsymbol{f}^{(t+1)} = \arg\min_{\boldsymbol{f}'\geq 0} Q(\boldsymbol{f}', \boldsymbol{f}^{(t)})$$

This implies that for any other feasible flow, including $\boldsymbol{f}^{(t)}$ itself, we have $Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)}) \leq Q(\boldsymbol{f}^{(t)}, \boldsymbol{f}^{(t)})$.

Now, let's evaluate $Q(\boldsymbol{f}^{(t)}, \boldsymbol{f}^{(t)})$. When $\boldsymbol{f}' = \boldsymbol{f}^{(t)}$, the change in flow is zero ($f'_{ij,k} - f^{(t)}_{ij,k} = 0$). Substituting this into the definition of Q yields:

$$Q(\boldsymbol{f}^{(t)}, \boldsymbol{f}^{(t)}) = \frac{1}{2} \sum_{(i,j)\in E} \left[\left(\sum_{k\in\mathcal{K}} f^{(t)}_{ij,k} + r^{(t)}_{ij} - u_{ij} \right)^2 + \sum_{k\in\mathcal{K}} (h^{(t)}_{ik})^2 + \sum_{k\in\mathcal{K}} (h^{(t)}_{jk})^2 \right]$$

This is identical to the definition of the global objective function $z(\boldsymbol{f}^{(t)})$ in Expression (12). Thus, we have $Q(\boldsymbol{f}^{(t)}, \boldsymbol{f}^{(t)}) = z(\boldsymbol{f}^{(t)})$. Combining these facts gives us our first key inequality:

$$Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)}) \leq z(\boldsymbol{f}^{(t)})$$

Step 2: Proving that $z(\boldsymbol{f}^{(t+1)}) \leq Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)})$

Next, we compare the true global objective value $z(\boldsymbol{f}^{(t+1)})$ under the new flow with the value of the surrogate function $Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)})$.

$$z(\boldsymbol{f}^{(t+1)}) = \frac{1}{2} \sum_{(i,j)\in E} \left[(\psi^{(t+1)}_{ij})^2 + \sum_{k\in\mathcal{K}} (h^{(t+1)}_{ik})^2 + \sum_{k\in\mathcal{K}} (h^{(t+1)}_{jk})^2 \right]$$

$$Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)}) = \frac{1}{2} \sum_{(i,j)\in E} \left[\psi^{(t+1)2}_{ij} + \sum_{k\in\mathcal{K}} (h^{(t)}_{ik} - \Delta f_{ij,k})^2 + \sum_{k\in\mathcal{K}} (h^{(t)}_{jk} + \Delta f_{ij,k})^2 \right]$$

where $\Delta f_{ij,k} = f^{(t+1)}_{ij,k} - f^{(t)}_{ij,k}$.

Comparing the two expressions, the congestion terms $(\psi^{(t+1)}_{ij})^2$ are identical. We only need to compare the height terms. The height at vertex i for commodity k under the new flow $\boldsymbol{f}^{(t+1)}$ is given by:

$$h^{(t+1)}_{ik} = h^{(t)}_{ik} + \frac{1}{\delta_i} \left(\sum_{j\in\delta^-(i)} \Delta f_{ji,k} - \sum_{j\in\delta^+(i)} \Delta f_{ij,k} \right).$$

To apply Jensen's inequality, we rewrite $h^{(t+1)}_{ik}$ as an average by distributing $h^{(t)}_{ik} = \frac{1}{\delta_i} \sum_{j\in\delta(i)} h^{(t)}_{ik}$:

$$h^{(t+1)}_{ik} = \frac{1}{\delta_i} \left(\sum_{j\in\delta^+(i)} (h^{(t)}_{ik} - \Delta f_{ij,k}) + \sum_{j\in\delta^-(i)} (h^{(t)}_{ik} + \Delta f_{ji,k}) \right)$$

Since the function x^2 is convex, we can apply Jensen's inequality, which states that the square of an average is less than or equal to the average of the squares:

$$(h^{(t+1)}_{ik})^2 \leq \frac{1}{\delta_i} \left(\sum_{j\in\delta^+(i)} (h^{(t)}_{ik} - \Delta f_{ij,k})^2 + \sum_{j\in\delta^-(i)} (h^{(t)}_{ik} + \Delta f_{ji,k})^2 \right)$$

Multiplying by δ_i and summing over all vertices i and commodities k:

$$\sum_{i\in V, k\in K} \delta_i (h^{(t+1)}_{ik})^2 \leq \sum_{i\in V, k\in K} \left(\sum_{j\in\delta^+(i)} (h^{(t)}_{ik} - \Delta f_{ij,k})^2 + \sum_{j\in\delta^-(i)} (h^{(t)}_{ik} + \Delta f_{ji,k})^2 \right)$$

The left-hand side (LHS) is the height-energy component of $2 \cdot z(\boldsymbol{f}^{(t+1)})$. We can rearrange the right-hand side (RHS) by summing over edges (i, j) instead of vertices. Each edge (i, j) contributes exactly two terms to the sum: one for vertex i (as an outgoing edge) and one for vertex j (as an incoming edge). This gives:

$$\text{RHS} = \sum_{(i,j)\in E} \sum_{k\in K} \left[(h_{ik}^{(t)} - \Delta f_{ij,k})^2 + (h_{jk}^{(t)} + \Delta f_{ij,k})^2 \right]$$

This is precisely the height-energy component of $2 \cdot Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)})$. Since the congestion terms in $z(\boldsymbol{f}^{(t+1)})$ and $Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)})$ are identical, this establishes our second key inequality:

$$z(\boldsymbol{f}^{(t+1)}) \leq Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)})$$

Combining our two inequalities, we have:

$$z(\boldsymbol{f}^{(t+1)}) \leq Q(\boldsymbol{f}^{(t+1)}, \boldsymbol{f}^{(t)}) \leq z(\boldsymbol{f}^{(t)})$$

This proves that the sequence of global objective function values $\{z(\boldsymbol{f}^{(t)})\}$ is monotonically non-increasing. Since $z(\boldsymbol{f}) \geq 0$, the sequence is bounded below. By the Monotone Convergence Theorem, the sequence must converge to a limit. This implies that the flow vector $\boldsymbol{f}^{(t)}$ converges to a fixed point, which, by its construction as a solution to the KKT conditions, is a stable pseudo-flow.

□

4.4 Exact Solution of the Edge Subproblem

The effectiveness of the PDR algorithm hinges on our ability to solve the edge subproblem (Expression 14) efficiently and exactly. While it is a Quadratic Program (QP), its specific structure allows for a closed-form analytical solution. This section derives this solution by applying the Karush-Kuhn-Tucker (KKT) conditions, demonstrating that the optimization for each edge reduces to a simple, fast computation.

First, let's restate the objective function for a single edge (i, j) at iteration t. We seek to find the next-iteration flows, which we'll denote as $f'_{ij,k}$ for clarity, that minimize z_{ij}:

$$\begin{aligned} \min_{f'_{ij,k}\geq 0, r'_{ij}\geq 0} \quad z_{ij} = & \frac{1}{2}\left(\sum_{k\in\mathcal{K}} f'_{ij,k} + r'_{ij} - u_{ij}\right)^2 \\ & + \frac{1}{2}\sum_{k\in\mathcal{K}} \left(h_{ik}^{(t)} + f_{ij,k}^{(t)} - f'_{ij,k}\right)^2 \\ & + \frac{1}{2}\sum_{k\in\mathcal{K}} \left(h_{jk}^{(t)} - f_{ij,k}^{(t)} + f'_{ij,k}\right)^2 \end{aligned}$$

This objective function is strictly convex and its constraints are linear, so the KKT conditions are both necessary and sufficient for optimality.

4.4.1 Karush-Kuhn-Tucker (KKT) Formulation

We introduce Lagrange multipliers $\lambda_k \geq 0$ for the non-negativity constraints $f'_{ij,k} \geq 0$ and $\eta \geq 0$ for $r'_{ij} \geq 0$. The Lagrangian $\mathcal{L}$ is:

$$\mathcal{L} = z_{ij} - \sum_{k\in\mathcal{K}} \lambda_k f'_{ij,k} - \eta r'_{ij}$$

The stationarity condition requires the gradient of $\mathcal{L}$ with respect to the primal variables ($f'_{ij,k}$ and r'_{ij}) to be zero.

1. **Derivative with respect to r'_{ij}:** Let $\psi'_{ij} = \sum_{k\in\mathcal{K}} f'_{ij,k} + r'_{ij} - u_{ij}$ be the new congestion value.

$$\frac{\partial\mathcal{L}}{\partial r'_{ij}} = \left(\sum_{k\in\mathcal{K}} f'_{ij,k} + r'_{ij} - u_{ij}\right) - \eta = \psi'_{ij} - \eta = 0 \implies \boldsymbol{\psi'_{ij} = \eta}$$

This provides a direct physical interpretation: the optimal congestion level ψ'_{ij} is equal to the dual variable η associated with the slack capacity.

2. **Derivative with respect to $f'_{ij,k}$:**

$$\frac{\partial \mathcal{L}}{\partial f'_{ij,k}} = \psi'_{ij} - (h^{(t)}_{ik} + f^{(t)}_{ij,k} - f'_{ij,k}) + (h^{(t)}_{jk} - f^{(t)}_{ij,k} + f'_{ij,k}) - \lambda_k = 0$$

Rearranging the terms to solve for $f'_{ij,k}$:

$$\psi'_{ij} + 2f'_{ij,k} - (h^{(t)}_{ik} - h^{(t)}_{jk} + 2f^{(t)}_{ij,k}) - \lambda_k = 0$$

To simplify, we define a constant "driving potential" c_k for each commodity k, which combines all known values from iteration t:

$$c_k \equiv \frac{1}{2}(h^{(t)}_{ik} - h^{(t)}_{jk} + 2f^{(t)}_{ij,k})$$

The stationarity condition then simplifies to $2f'_{ij,k} = 2c_k - \psi'_{ij} + \lambda_k$.

4.4.2 Deriving the Flow Update Rule

The complementary slackness condition, $\lambda_k f'_{ij,k} = 0$, allows us to determine the optimal flow $f'_{ij,k}$:

- If $f'_{ij,k} > 0$: Complementary slackness requires $\lambda_k = 0$. The flow is $f'_{ij,k} = c_k - \frac{\psi'_{ij}}{2}$. This is only valid if $2c_k > \psi'_{ij}$.
- If $f'_{ij,k} = 0$: $\lambda_k \geq 0$ is possible. The stationarity condition gives $\lambda_k = \psi'_{ij} - 2c_k \geq 0$, which means this case holds if $2c_k \leq \psi'_{ij}$.

Combining these two outcomes gives a single, elegant expression for the optimal flow as a function of the yet-unknown congestion level ψ'_{ij}:

$$f'_{ij,k} = \max\left(0, c_k - \frac{\psi'_{ij}}{2}\right) \tag{15}$$

The problem has been reduced to finding a single scalar value, ψ'_{ij}.

4.4.3 Solving for the Congestion Level ψ'_{ij}

The final KKT condition, $\eta r'_{ij} = 0$, combined with our finding that $\eta = \psi'_{ij}$, gives $\psi'_{ij} r'_{ij} = 0$. This implies that either the congestion is zero or the slack capacity is zero, leading to two distinct cases.

Case 1: Uncongested Edge ($r'_{ij} > 0$) If the optimal solution has unused capacity, then we must have $\boldsymbol{\psi'_{ij} = 0}$. Substituting this into Equation (15) gives the candidate solution:

$$f'_{ij,k} = \max(0, c_k)$$

This solution is valid if and only if the resulting total flow is less than the capacity, which is the condition for $r'_{ij} > 0$ to be possible. We check this condition:

$$\text{If} \quad \sum_{k \in \mathcal{K}} \max(0, c_k) < u_{ij}, \quad \text{then this is the optimal solution.}$$

Case 2: Congested Edge ($r'_{ij} = 0$) If the check for Case 1 fails (i.e., $\sum_k \max(0, c_k) \geq u_{ij}$), the edge must be at or over capacity, meaning the optimal slack is $\boldsymbol{r'_{ij} = 0}$. In this case, the congestion ψ'_{ij} can be positive.

With $r'_{ij} = 0$, the definition of ψ'_{ij} becomes $\psi'_{ij} = \sum_k f'_{ij,k} - u_{ij}$. We find the value of ψ'_{ij} by substituting our expression for $f'_{ij,k}$ (Equation 15) into this definition:

$$\psi'_{ij} = \sum_{k \in \mathcal{K}} \max\left(0, c_k - \frac{\psi'_{ij}}{2}\right) - u_{ij}$$

The right-hand side of this equation is a piecewise-linear, continuous, and monotonically decreasing function of ψ'_{ij}. We are looking for the unique root of this equation. This root can be found very efficiently by first sorting the potential values c_k. The breakpoints of the function occur at $\psi'_{ij} = 2c_k$. By iterating through the sorted list of potentials, we can quickly identify the linear segment that contains the root. This procedure takes $O(K \log K)$ time, dominated by sorting the K commodity potentials.

Once the unique root $\psi'_{ij} \geq 0$ is found, the final optimal flows are calculated using Equation (15). The complete algorithm for this procedure is detailed in Algorithm 2.

Algorithm 2 Solver for the Edge Subproblem

```
Require: Current vertex heights and flows $\{h^t_{ik}, h^t_{jk}, f^t_{ij,k}\}_{k\in\mathcal{K}}$, and edge capacity $u_{ij}$.
Ensure: Optimal next-iteration flows $f^{t+1}_{ij,k}$, slack capacity $r_{ij}$, and new congestion $\psi_{ij}$.
 1: 1. Preprocessing:
 2: For each commodity $k \in \mathcal{K}$, compute its potential:
        $c_k \leftarrow \frac{1}{2}\Big((h^t_{ik} + f^t_{ij,k}) - (h^t_{jk} - f^t_{ij,k})\Big)$.
 3: 2. Check for Congestion:
 4: Compute the potential flow sum $s \leftarrow \sum_{k\in\mathcal{K}} \max(0, c_k)$.
 5: if $s < u_{ij}$ then                                    ▷ Case 1: The edge is not congested.
 6:     $r_{ij} \leftarrow u_{ij} - s$.
 7:     $\psi_{ij} \leftarrow 0$.
 8: else                                 ▷ Case 2: The edge is congested, solve for $\psi_{ij}$.
 9:     $r_{ij} \leftarrow 0$.
10:     Let $\mathcal{K}^+ = \{k \in \mathcal{K} \mid c_k > 0\}$.
11:     Sort potentials in $\mathcal{K}^+$ in descending order: $c_{(1)} \geq c_{(2)} \geq \cdots \geq c_{(m)}$.
12:     sum_c $\leftarrow 0$.
13:     for $p = 1, \ldots, m$ do
14:         sum_c $\leftarrow$ sum_c $+ c_{(p)}$.
15:         $\psi_{\text{candidate}} \leftarrow (\text{sum\_c} - u_{ij})/(1 + p/2)$.
16:                     ▷ A candidate solution is valid only if it falls within the assumed interval.
17:         Let $c_{\text{upper\_bound}} = c_{(p)}$.
18:         Let $c_{\text{lower\_bound}} = (p < m) ? c_{(p+1)} : 0$.
19:         if $2 \cdot c_{\text{lower\_bound}} < \psi_{\text{candidate}} \leq 2 \cdot c_{\text{upper\_bound}}$ then
20:             $\psi_{ij} \leftarrow \psi_{\text{candidate}}$.                ▷ This is the unique, correct root.
21:             break                                                         ▷ Exit the loop.
22:         end if
23:     end for
24: end if
25: 3. Final Flow Calculation:
26: For each commodity $k \in \mathcal{K}$:
        $f^{t+1}_{ij,k} \leftarrow \max(0, c_k - \psi_{ij}/2)$.
27: return $(f^{t+1}_{ij,k})_{k\in\mathcal{K}}$, $r_{ij}$, and $\psi_{ij}$.
```

5 PDR via Heuristic Algorithm

While the PDR algorithm with an exact subproblem solver guarantees convergence, its practical performance can be limited when the number of commodities, $|\mathcal{K}|$, is very large. The efficiency of the exact solver (Algorithm 2) depends on sorting the commodity potentials, an operation with a computational complexity of $O(|\mathcal{K}| \log |\mathcal{K}|)$ for each edge. This sorting step can become a significant computational bottleneck in large-scale scenarios. To address this challenge, we propose more computationally efficient heuristic variants based on the **gradient projection** method.

5.1 Adaptive Gradient Descent Algorithm

This approach leverages a key insight from our formulation: as noted in Remark 2, the negative gradient of the global objective function z with respect to the flow $f_{ij,k}$ is precisely the potential difference, $\phi_{ij,k}$. The potential difference therefore indicates the direction of steepest descent for the objective function. This naturally leads to a gradient descent heuristic, which adjusts the flow on each edge in proportion to this gradient.

The core of the algorithm lies in the flow update rule. For each edge (i, j) and commodity k, a proposed flow change is computed based on the potential difference and a per-edge **learning rate** β_{ij}. This change is then **projected** onto the feasible set to ensure $f_{ij,k}^{(t+1)} \geq 0$:

$$\phi_{ij,k}^{(t)} = (h_{ik}^{(t)} - h_{jk}^{(t)} - \psi_{ij}^{(t)})$$

$$\Delta f_{ij,k} \leftarrow \max(-f_{ij,k}^{(t)}, \beta_{ij} \cdot \phi_{ij,k}^{(t)})$$

Before applying an update, the algorithm first evaluates whether the step $\{\Delta f_{ij,k}\}_{k\in\mathcal{K}}$ for an edge (i,j) will decrease the **local objective function** z_{ij} of Expression 13. If the update would lead to an increase in the objective value, the step for that edge is rejected for the current iteration. This acceptance/rejection criterion ensures that the objective function is monotonically non-increasing, which is a key property for ensuring convergence.

The learning rate β_{ij} for each edge is adapted dynamically. If a step is rejected, it indicates the learning rate was too aggressive, so it is multiplicatively decreased. Conversely, the algorithm periodically attempts to increase the learning rate to accelerate progress. This adaptive scheme allows the algorithm to be both aggressive in its search for the minimum and cautious when necessary, improving overall performance.

The complete procedure, outlined in Algorithm 3, provides a computationally inexpensive alternative to the exact subproblem solver. It trades guaranteed per-iteration optimality for extremely fast, scalable iterations, making it particularly well-suited for problems with a very large number of commodities.

Algorithm 3 PDR via Adaptive Gradient Descent

```
1: Input: Graph $G(V,E)$, capacities $\{u_{ij}\}$, commodities $\{(s_k,t_k,d_k)\}$, tolerance $\epsilon_{\text{obj}} > 0$.
2: Parameters: Initial learning rate $\beta_{init}$, min/max rates $\beta_{min}, \beta_{max}$, factors $\alpha_{inc} > 1$, $\alpha_{dec} < 1$.
3: Initialize: Set initial flow $f^{(0)} \leftarrow 0$, iteration $t \leftarrow 0$, learning rate $\beta_{ij} \leftarrow \beta_{init}, \forall(i,j) \in E$, previous objective value
   $z_{\text{prev}} \leftarrow \infty$.
4: while $t <$ max_iterations do
5:     for all vertices $i \in V$, commodities $k \in \mathcal{K}$ in parallel do
6:         Flow imbalance $\Delta_{ik}^{(t)} \leftarrow \sum_{j\in\delta^-(i)} f_{ji,k}^{(t)} - \sum_{j\in\delta^+(i)} f_{ij,k}^{(t)} + \hat{\Delta}_{ik}$
7:         Vertex height $h_{ik}^{(t)} \leftarrow \Delta_{ik}^{(t)}/\delta_i$
8:     end for
9:     for all edges $(i,j) \in E$ in parallel do
10:        Edge congestion $\psi_{ij}^{(t)} \leftarrow \max\left(0, \sum_{k\in\mathcal{K}} f_{ij,k}^{(t)} - u_{ij}\right)$
11:    end for
12:    $z^{(t)} \leftarrow \frac{1}{2}\sum_{(i,j)\in E}(\psi_{ij}^{(t)})^2 + \frac{1}{2}\sum_{i\in V,k\in\mathcal{K}} \delta_i(h_{ik}^{(t)})^2$
13:    if $|z^{(t)} - z_{\text{prev}}| < \epsilon_{\text{obj}}$ then
14:        break
15:    end if
16:    $z_{\text{prev}} \leftarrow z^{(t)}$
17:                                                    ▷ Update flows using a projected gradient step
18:    for all edges $(i,j) \in E$ in parallel do
19:        For each commodity $k \in \mathcal{K}$, $\phi_{ij,k} \leftarrow h_{ik}^{(t)} - h_{jk}^{(t)} - \psi_{ij}^{(t)}$.
20:        For each commodity $k \in \mathcal{K}$, $\Delta f_{ij,k} \leftarrow \max(-f_{ij,k}^{(t)}, \beta_{ij} \cdot \phi_{ij,k})$.
21:        Estimate objective change $\Delta z_{ij}$ if update $\{\Delta f_{ij,k}\}$ is applied.
22:        if $\Delta z_{ij} \leq 0$ then
23:            For each commodity $k \in \mathcal{K}$, $f_{ij,k}^{(t+1)} \leftarrow f_{ij,k}^{(t)} + \Delta f_{ij,k}$.
24:        else                                        ▷ Reject update and decrease learning rate
25:            For each commodity $k \in \mathcal{K}$, $f_{ij,k}^{(t+1)} \leftarrow f_{ij,k}^{(t)}$.
26:            $\beta_{ij} \leftarrow \max(\beta_{min}, \beta_{ij} \cdot \alpha_{dec})$.
27:        end if
28:        if $t \pmod{10} = 0$ then
29:            $\beta_e \leftarrow \beta_e \cdot \alpha_{inc}$           ▷ Periodically increase learning rate
30:        end if
31:    end for
32:    $t \leftarrow t + 1$
33: end while
34: Return final flow $f^{(t)}$
```

5.2 Momentum-Based Adaptive Gradient Descent Algorithm

While the Adaptive Gradient Descent method provides a significant computational advantage, there are numerous techniques to accelerate gradient-based methods. Here, we introduce a simple yet powerful and widely-used approach: incorporating **momentum**. The resulting method, the Momentum-Based Adaptive Gradient Descent algorithm, is detailed in Algorithm 4.

To implement this, we introduce a **velocity vector**, $\boldsymbol{v}$, for each commodity on each edge. This vector accumulates a history of the gradient-based steps, effectively smoothing the optimization trajectory. As shown in Algorithm 4, the update process is modified as follows. First, the standard projected step, $\Delta f_{ij,k}$, is calculated as in the previous method. Then, the velocity is updated by blending its previous value with the new step, controlled by a momentum coefficient, $\gamma \in [0, 1)$:

$$v_{ij,k}^{(t+1)} \leftarrow \gamma \cdot v_{ij,k}^{(t)} + \Delta f_{ij,k}$$

The flow is then updated by applying this new velocity, and the result is projected to ensure non-negativity: $f_{ij,k}^{(t+1)} \leftarrow \max(0, f_{ij,k}^{(t)} + v_{ij,k}^{(t+1)})$.

Note that Algorithm 4 is identical to Algorithm 3, with the exception of the modifications highlighted in blue. Numerical experiments show that such a small modification can significantly accelerate convergence.

Algorithm 4 PDR via Gradient Descent with Momentum

```
1: Input: Graph $G(V, E)$, capacities $\{u_{ij}\}$, commodities $\{(s_k, t_k, d_k)\}$, tolerance $\epsilon_{\text{obj}} > 0$.
2: Parameters: Initial learning rate $\beta_{init}$, min/max rates $\beta_{min}, \beta_{max}$, factors $\alpha_{inc} > 1$, $\alpha_{dec} < 1$, momentum coefficient $\gamma \in [0, 1)$.
3: Initialize: Set initial flow $f^{(0)} \leftarrow 0$, velocity $v^{(0)} \leftarrow 0$, iteration $t \leftarrow 0$, learning rate $\beta_{ij} \leftarrow \beta_{init}, \forall (i, j) \in E$.
4: while $t <$ max_iterations do
5:     for all vertices $i \in V$, commodities $k \in \mathcal{K}$ in parallel do
6:         Flow imbalance $\Delta_{ik}^{(t)} \leftarrow \sum_{j\in\delta^-(i)} f_{ji,k}^{(t)} - \sum_{j\in\delta^+(i)} f_{ij,k}^{(t)} + \hat{\Delta}_{ik}$
7:         Vertex height $h_{ik}^{(t)} \leftarrow \Delta_{ik}^{(t)}/\delta_i$
8:     end for
9:     for all edges $(i, j) \in E$ in parallel do
10:        Edge congestion $\psi_{ij}^{(t)} \leftarrow \max\left(0, \sum_{k\in\mathcal{K}} f_{ij,k}^{(t)} - u_{ij}\right)$
11:    end for
12:    $z^{(t)} \leftarrow \frac{1}{2}\sum_{(i,j)\in E}(\psi_{ij}^{(t)})^2 + \frac{1}{2}\sum_{i\in V,k\in\mathcal{K}} \delta_i (h_{ik}^{(t)})^2$
13:    if $|z^{(t)} - z_{\text{prev}}| < \epsilon_{\text{obj}}$ then
14:        break
15:    end if
16:    $z_{\text{prev}} \leftarrow z^{(t)}$
17:    for all edges $(i, j) \in E$ in parallel do
18:        For each commodity $k \in \mathcal{K}$, $\phi_{ij,k} \leftarrow h_{ik}^{(t)} - h_{jk}^{(t)} - \psi_{ij}^{(t)}$.
19:        For each commodity $k \in \mathcal{K}$, $\Delta f_{ij,k} \leftarrow \max(-f_{ij,k}^{(t)}, \beta_{ij} \cdot \phi_{ij,k})$.
20:        Estimate objective change $\Delta z_{ij}$ if update $\{\Delta f_{ij,k}\}$ is applied.
21:        if $\Delta z_{ij} \leq 0$ then
22:            For each $k \in \mathcal{K}$, $v_{ij,k}^{(t+1)} \leftarrow \gamma \cdot v_{ij,k}^{(t)} + \Delta f_{ij,k}$.
23:            For each commodity $k \in \mathcal{K}$, $f_{ij,k}^{(t+1)} \leftarrow \max(0, f_{ij,k}^{(t)} + v_{ij,k}^{(t+1)})$.
24:        else
25:            For each commodity $k \in \mathcal{K}$, $f_{ij,k}^{(t+1)} \leftarrow f_{ij,k}^{(t)}$, $v_{ij,k}^{(t+1)} \leftarrow v_{ij,k}^{(t)}$.
26:            $\beta_{ij} \leftarrow \max(\beta_{min}, \beta_{ij} \cdot \alpha_{dec})$.
27:        end if
28:        if $t \pmod{10} = 0$ then
29:            $\beta_{ij} \leftarrow \beta_{ij} \cdot \alpha_{inc}$
30:        end if
31:    end for
32:    $t \leftarrow t + 1$
33: end while
34: Return final flow $f^{(t)}$
```

6 Further Discussion on Optimality Condition

As commonly understood, directed edges correspond to pipes of the physical network and vertices are pipe junctions. In practice, for any physical network, it is impossible for the flow in a pipe to exceed its capacity . In this section we would discuss the optimality condition for multicommodity flow problem when only relaxing the flow conservation constraints.

The basic formulation is as follows:

$$\begin{aligned} \textbf{min} \quad & z = \sum_{i,k} \int_0^{\Delta_{ik}} h_{ik}(\omega) d\omega \\ \textbf{s.t} \quad & \sum_{k \in \mathcal{K}} f_{ij,k} \leq u_{ij}, \forall (i,j) \in E \\ & f_{ij,k} \geq 0, \forall k \in \mathcal{K}, (i,j) \in E \end{aligned} \tag{16}$$

where h_{ik} is the height function and Δ_{ik} is defined as Expression (2).

In the programming above, the objective function is the sum of the integrals of the vertex height functions. The flow conservation constraints are relaxed but the capacity constraints are maintained. Obviously, the feasible region of Expression (1) is not empty if and only if the minimum value of the objective function of Programming (16) is zero; the feasible region of Expression (1) is empty if and only if the minimum value of the objective function of Programming (16) is greater than zero. Since Programming (16) is convex, the solution $\boldsymbol{f}^*$ of Programming (16) is optimal if and only if it satisfies the following Karush-Kuhn-Tucker conditions:

$$\begin{aligned} & \textbf{Stationarity} \\ & \quad -\frac{\partial z}{\partial f_{ij,k}} = -\mu_{ij,k} + \gamma_{ij}, \forall k \in \mathcal{K}, (i,j) \in E \\ & \textbf{Primal feasibility} \\ & \quad \sum_{k \in \mathcal{K}} f_{ij,k} \leq u_{ij}, \forall (i,j) \in E \\ & \quad -f_{ij,k} \leq 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & \textbf{Dual feasibility} \\ & \quad \mu_{ij,k} \geq 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & \quad \gamma_{ij} \geq 0, \forall (i,j) \in E \\ & \textbf{Complementary slackness} \\ & \quad \mu_{ij,k} f_{ij,k} = 0, \forall k \in \mathcal{K}, (i,j) \in E \\ & \quad \gamma_{ij} (\sum_{k \in \mathcal{K}} f_{ij,k} - u_{ij}) = 0, \forall (i,j) \in E. \end{aligned} \tag{17}$$

Obviously,

$$\begin{aligned} \frac{\partial z}{\partial f_{ij,k}} &= h_{ik}(\Delta_{ik}) \frac{\partial \Delta_{ik}}{\partial f_{ij,k}} + h_{jk}(\Delta_{jk}) \frac{\partial \Delta_{jk}}{\partial f_{ij,k}} \\ &= (-h_{ik}(\Delta_{ik})) + (h_{jk}(\Delta_{jk})) \\ &= h_{jk} - h_{ik}. \end{aligned} \tag{18}$$

Substituting the expression above into Stationarity expression in KKT conditions,

$$\gamma_{ij} = \mu_{ij,k} + h_{ik} - h_{jk}, \forall k \in \mathcal{K}, (i,j) \in E.$$

If edge(i,j) is not saturated, by complementary slackness, $\gamma_{ij} = 0$. If edge(i,j) is saturated, $\gamma_{ij} \geq 0$. Intuitively, γ_{ij} could be interpreted as the pipe pressure on edge(i,j). If edge(i,j) is not saturated, the *pipe pressure* is zero; otherwise it is greater than or equal to zero.

For any used edge of commodity k, i.e. $f_{ij,k} > 0$, by complementary slackness $\mu_{ij,k} f_{ij,k} = 0$ in KKT conditions, we have $\mu_{ij,k} = 0$. Therefore,

$$\gamma_{ij} = h_{ik} - h_{jk}, \forall k \in \mathcal{K}, (i,j) \in E.$$

That is, the height difference between vertex i and vertex j for commodity k is equal to the pipe pressure γ_{ij} of edge (i, j).

That is, for any commodity k using edge (i, j), its height difference must equal the pipe pressure γ_{ij}. A critical consequence of this result is that *for any saturated edge, the height difference $h_{ik} - h_{jk}$ must be the same for all commodities k that have a positive flow on that edge.*

For any unused edge of commodity k, due to $\mu_{ij,k} \geq 0$, we have

$$\gamma_{ij} = \mu_{ij,k} + h_{ik} - h_{jk} \geq h_{ik} - h_{jk}, \\ \forall k \in \mathcal{K}, (i, j) \in E$$

That is, the height difference between vertex i and vertex j for commodity k is less than or equal to the pipe pressure γ_{ij}.

Note that the pipe pressure γ_{ij} is zero when the edge is unsaturated. Therefore, the optimality conditions can be concluded as follows:

(i) for any used edge of commodity k, if it is unsaturated, the height difference between vertex i and vertex j for commodity k is zero;

(ii) for any used edge of commodity k, if it is saturated, the height difference between vertex i and vertex j for commodity k is equal to the pipe pressure γ_{ij};

(iii) for any unused edge of commodity k, the height difference between vertex i and vertex j for commodity k is less than or equal to the pipe pressure γ_{ij}.

7 Numerical Results

In this section, we empirically evaluate the performance of the proposed algorithms on a suite of standard benchmark problems. We first describe the test instances and then present a comparative analysis of the results.

7.1 Test Problems

Our numerical experiments were conducted on three sets of well-known benchmark problems for the multicommodity flow problem(Babonneau et al., 2006; Larsson and Yuan, 2004).

1. **Planar Problems:** This set, generated by Larsson and Yuan (2004), comprises 10 instances designed to simulate telecommunication networks. Vertices are placed randomly in a plane, and arcs connect neighboring vertices to form a planar graph. Commodities, demands, and capacities are chosen randomly from uniform distributions.
2. **Grid Problems:** This set contains 15 instances with a grid network structure, where most vertices have a degree of four. This topology is characterized by a large number of alternative paths between vertex pairs.
3. **Telecommunication Problems:** This collection includes real-world and realistic problems of varying scales. Notably, it features the `904` instance, a large-scale network with 904 arcs and 11,130 commodities, as referenced in the survey by Ouorou et al. (2000).

These problem sets are publicly available[1] and standard for benchmarking multicommodity flow algorithms. Since the objective of our study is to find a *feasible* flow (i.e., a zero-stable pseudo-flow), the arc cost data present in the original benchmarks were *disregarded*. For each problem, we report the number of vertices $|V|$, arcs $|E|$, and commodities $|\mathcal{K}|$.

7.2 Numerical Experiments

We implemented and compared the three primary algorithmic variants discussed in this paper:

- `PDR-ESO` **(PDR via Exact Subproblem Optimization):** The method from Algorithm 1, which solves the edge subproblem (Expression (13)) exactly in each iteration.
- `PDR-AGD` **(PDR via Adaptive Gradient Descent):** The heuristic method from Algorithm 3, which uses a projected gradient step with an adaptive learning rate.
- `PDR-GDM` **(PDR via Gradient Descent with Momentum):** The enhanced heuristic from Algorithm 4, which incorporates a momentum term to accelerate and stabilize convergence.

[1] `http://www.di.unipi.it/di/groups/optimize/Data/MMCF.html`

7.2.1 Experimental Setup

All algorithms were implemented in CUDA and executed on a NVIDIA A100 GPU. The algorithms were tested on 28 benchmark instances, with performance primarily evaluated based on the number of iterations and total computation time required to satisfy the convergence criterion $|z^{(t)} - z^{(t-1)}| < 10^{-6}$. Here, $z^{(t)}$ is the objective value (from Expression (12), representing the sum of squared constraint violations) at iteration t.

For PDR-AGD, the parameters were configured as follows: the initial learning rate β_{init} was set to 0.25, with a minimum learning rate $\beta_{\text{min}} = 1.0/|\mathcal{K}|$ and a maximum $\beta_{\text{max}} = 0.5$. The learning rate adjustment factors were $\alpha_{\text{inc}} = 2.0$ and $\alpha_{\text{dec}} = 0.5$.

For PDR-GDM, the parameters were set with an initial learning rate $\beta_{\text{init}} = 0.1$, momentum coefficient $\gamma = 0.9$, a minimum $\beta_{\text{min}} = 1.0/|\mathcal{K}|$, and a maximum $\beta_{\text{max}} = 0.2$. The adjustment factors α_{inc} and α_{dec} remained 2.0 and 0.5, respectively. The complete experimental results are presented in Table 1.

Table 2: Complete performance comparison of PDR-ESO, PDR-AGD, and PDR-GDM algorithms.

				PDR-ESO Algorithm			**PDR-AGD Algorithm**			**PDR-GDM Algorithm**		
Instance	$\lvert E\rvert$	$\lvert V\rvert$	$\lvert\mathcal{K}\rvert$	**Iter.**	**Time (s)**	**Obj z**	**Iter.**	**Time (s)**	**Obj z**	**Iter.**	**Time (s)**	**Obj z**
C22	22	14	23	56	0.0064	0.000 002 3	61	0.0053	0.000 002 8	115	0.0102	0.000 019 5
C148	148	61	122	1075	0.1451	0.000 133 1	1734	0.1668	0.001 022 9	399	0.0389	0.001 454 6
C904	904	106	11 107	120	0.1476	0.000 006 1	126	0.0670	0.000 005 8	175	0.0959	0.000 020 8
Cgd1	80	25	50	374	0.0458	0.000 020 3	441	0.0418	0.000 024 5	176	0.0190	0.000 060 4
Cgd2	80	25	100	1915	0.2586	0.000 164 8	7282	0.6536	0.001 270 1	212	0.0193	0.000 008 3
Cgd3	360	100	50	1143	0.1595	0.000 068 5	1168	0.1145	0.000 070 5	190	0.0187	0.000 016 9
Cgd4	360	100	100	2108	0.3106	0.000 169 2	2615	0.2521	0.000 224 5	820	0.0798	0.000 107 5
Cgd5	840	225	100	2523	0.3990	0.000 161 1	2536	0.2556	0.000 163 3	622	0.0658	0.000 036 5
Cgd6	840	225	200	3484	0.6091	0.000 399 6	5295	0.5536	0.000 535 8	1391	0.1442	0.003 731 3
Cgd7	1520	400	400	4595	1.8731	0.000 296 2	4610	0.5137	0.000 298 3	1243	0.1448	0.000 089 6
Cgd8	2400	625	500	7273	4.1351	0.000 508 9	7893	0.9848	0.000 600 5	1814	0.2363	0.005 035 1
Cgd9	2400	625	1000	6481	4.1993	0.016 942 7	12 614	1.9144	0.001 371 3	2534	0.4277	0.002 324 5
Cgd10	2400	625	2000	7820	6.2245	0.000 629 0	8742	2.2619	0.000 659 4	2453	0.6809	0.009 936 7
Cgd11	2400	625	4000	7954	6.8848	0.000 486 2	7963	3.4184	0.000 485 7	2057	0.9748	0.000 112 5
Cgd12	3480	900	6000	11 318	34.0878	0.000 726 5	11 338	9.0589	0.000 716 5	2977	2.6477	0.000 167 0
Cgd13	3480	900	12 000	11 803	55.6226	0.000 747 0	11 822	16.8680	0.000 731 7	3092	5.0515	0.000 169 1
Cgd14	4760	1225	16 000	15 864	118.0360	0.001 034 8	15 872	39.6654	0.001 023 2	4192	11.9781	0.000 236 0
Cgd15	4760	1225	32 000	16 470	240.2010	0.001 182 2	16 520	80.6890	0.001 095 6	4352	24.4519	0.000 239 4
Cpl30	150	30	92	340	0.0473	0.000 022 9	359	0.0325	0.000 024 4	140	0.0132	0.000 112 0
Cpl50	250	50	267	669	0.1315	0.000 049 6	880	0.0856	0.000 079 0	173	0.0174	0.000 009 4
Cpl80	440	80	543	620	0.1767	0.000 307 2	1040	0.1092	0.000 109 4	202	0.0212	0.000 013 3
Cpl100	532	100	1085	1048	0.3735	0.000 100 3	1362	0.1541	0.000 114 6	239	0.0278	0.000 009 5
Cpl150	850	150	2239	1735	1.1987	0.005 565 8	37 873	4.9982	0.028 710 0	1358	0.1977	0.000 012 0
Cpl300	1680	300	3584	2719	2.1256	0.000 207 1	2733	0.7969	0.000 207 4	682	0.2213	0.000 044 7
Cpl500	2842	500	3525	3222	3.0401	0.000 255 9	3228	1.4272	0.000 255 9	829	0.4028	0.000 057 6
Cpl800	4388	800	12 756	5345	35.7495	0.000 442 3	5473	10.7503	0.000 456 6	1358	3.0862	0.000 191 1
Cpl1000	5200	1000	20 026	9047	109.7860	0.003 745 6	10 695	38.5915	0.000 860 8	2298	9.4954	0.002 032 0
Cpl2500	12 990	2500	81 430	15 225	1845.3300	0.021 195 0	21 408	788.5840	0.001 843 0	4549	191.1520	0.005 032 1

7.2.2 Analysis of Convergence Behavior

The results clearly demonstrate the superior performance of the momentum-accelerated method. The PDR-GDM algorithm consistently converges in the fewest iterations and, consequently, the shortest time across the vast majority of instances. As a representative example, on instance Cpl1000, PDR-GDM converges in 2298 iterations, whereas the standard PDR-AGD method requires 10695 iterations—a 4.6-fold improvement.

An insightful and arguably counter-intuitive result is that the PDR-ESO algorithm, despite solving each edge subproblem to optimality, frequently requires a substantially greater number of iterations to converge than its approximation-based counterpart, PDR-GDM. This observation underscores the limitations of a purely greedy optimization approach. It leads to the pivotal conclusion that local optimality at each step does not guarantee, and may not even facilitate, rapid global convergence. Consequently, a sequence of less precise yet more aggressive steps, such as those generated by PDR-GDM, can be demonstrably more efficient, ultimately achieving convergence with a lower total iteration count.

7.2.3 Computational Time and Solution Quality

In terms of computational time, PDR-GDM **consistently emerges as the most efficient algorithm.** This superior performance is a direct result of its ability to significantly reduce the number of iterations while maintaining a computationally inexpensive update step. In contrast, PDR-ESO **is almost always the slowest**, hampered by the high computational cost of its exact solver ($O(|\mathcal{K}| \log |\mathcal{K}|)$ per edge). While PDR-AGD is generally faster than PDR-ESO, it cannot match the accelerated convergence of the momentum-based approach.

The practical impact of this efficiency is particularly evident on large-scale instances. For example, the Cgd15 **problem** involves **over 152 million** variables ($|E| \times |\mathcal{K}| = 4760 \times 32000$). Remarkably, PDR-GDM finds a feasible solution for this massive instance in just **24.45 seconds**.

Regarding solution quality, all three algorithms successfully reduce the objective value z—representing the sum of squared constraint violations—to a very small magnitude. It is important to contextualize this final objective value. Given the enormous number of variables and constraints in these benchmark problems, the final values of z indicate that the **violation per single constraint** (both for arc capacities and flow conservation) is **negligibly small**. Therefore, for all practical purposes, the solutions can be considered **fully feasible and are of high quality**, satisfying the primary goal of the study.

8 Conclusion

In this paper, we introduced a novel theoretical framework for the multicommodity flow problem, grounded in the physical intuition of potential (height) and congestion. By defining vertex-specific potential functions and edge-specific congestion functions, we established a simple and intuitive set of optimality conditions. We then reformulated the classical feasibility problem as a convex optimization problem with nonnegativity constraints, where the existence of a feasible solution is equivalent to the objective function attaining a minimum value of zero.

A key advantage of this framework is its edge-separable structure, which makes it particularly well-suited for parallel and distributed computation. Leveraging this property, we proposed three algorithms: a method that solves an exact quadratic subproblem for each edge, and two computationally efficient heuristics based on adaptive gradient descent and its momentum-accelerated variant. Numerical experiments on benchmark instances demonstrated that our algorithms are highly efficient for large-scale networks, with the momentum-based heuristic (PDR-GDM) exhibiting particularly rapid convergence.

While the proposed framework and algorithms show great promise for solving large-scale problems, this work also opens several avenues for future research. The most pressing theoretical question is the formal analysis of the worst-case convergence rates and computational complexities of our proposed algorithms. Furthermore, exploring more sophisticated adaptive learning rate and momentum schemes could lead to even faster convergence. Finally, applying this potential and congestion framework to other classes of network flow problems, such as those with nonlinear cost functions or additional side constraints, presents a rich area for further investigation.

9 Code and Data Disclosure

The code and data to support the numerical experiments in this paper can be found at https://github.com/mrgump-123/Localized-Multicommodity-Flow.

References

Baruch Awerbuch and Tom Leighton. A simple local-control approximation algorithm for multicommodity flow. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 459–468. IEEE, 1993.

Baruch Awerbuch and Tom Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 487–496, 1994.

Baruch Awerbuch, Rohit Khandekar, and Satish Rao. Distributed algorithms for multicommodity flow problems via approximate steepest descent framework. In *Symposium on Discrete Algorithms: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 7, pages 949–957, 2007.

Kyriakos Axiotis, Aleksander Mądry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 528–539. IEEE, 2022.

Frédéric Babonneau, Olivier du Merle, and J-P Vial. Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. *Operations Research*, 54(1):184–197, 2006.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Jan van Den Brand and Daniel J Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 493–502. IEEE, 2023.

Olivier Brun, Balakrishna Prabhu, and Josselin Vallet. A penalized best-response algorithm for nonlinear single-path routing problems. *Networks*, 69(1):52–66, 2017.

Minh N Bùi. A decomposition method for solving multicommodity network equilibria. *Operations Research Letters*, 50(1):40–44, 2022.

Jordi Castro and Antonio Frangioni. A parallel implementation of an interior-point algorithm for multicommodity network flows. In *International Conference on Vector and Parallel Processing*, pages 301–315. Springer, 2000.

Bala G Chandran and Dorit S Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Operations research*, 57(2):358–376, 2009.

Li Chen and Mingquan Ye. High-accuracy multicommodity flows via iterative refinement. *arXiv preprint arXiv:2304.11252*, 2023.

Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623. IEEE, 2022.

Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 273–282, 2011.

Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39, 2021.

Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 451–460, 2008.

Ming Ding, Rasmus Kyng, and Peng Zhang. Two-commodity flow is equivalent to linear programming under nearly-linear time reductions. *arXiv preprint arXiv:2201.11587*, 2022.

Daniele D'Agostino, Ivan Merelli, Marco Aldinucci, and Daniele Cesini. Hardware and software solutions for energy-efficient computing in scientific programming. *Scientific Programming*, 2021(1):5514284, 2021.

Noel Farrugia, Johann A Briffa, and Victor Buttigieg. Solving the multicommodity flow problem using an evolutionary routing algorithm in a computer network environment. *PloS one*, 18(4):e0278317, 2023.

Barak Fishbain, Dorit S Hochbaum, and Stefan Mueller. Competitive analysis of minimum-cut maximum flow algorithms in vision problems. *arXiv preprint arXiv:1007.4531*, 2010.

Lester Randolph Ford and Delbert Ray Fulkerson. Flows in networks. In *Flows in Networks*. Princeton university press, 2015.

Antonio Frangioni and Giorgio Gallo. A bundle type dual-ascent approach to linear multicommodity min-cost flow problems. *INFORMS Journal on Computing*, 11(4):370–393, 1999.

Yu Gao, Yang P Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 516–527. IEEE, 2022.

Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

Arthur M Geoffrion. Primal resource-directive approaches for optimizing nonlinear decomposable systems. *Operations Research*, 18(3):375–403, 1970.

Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, pages 81–90, 2015.

J-L Goffin, Jacek Gondzio, Robert Sarkissian, and J-P Vial. Solving nonlinear multicommodity flow problems by the analytic center cutting plane method. *Mathematical programming*, 76(1):131–154, 1997.

Andrew Goldberg and Robert Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 7–18, 1987.

Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.

Bernhard Haeupler, D Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. Low-step multicommodity flow emulators. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 71–82, 2024.

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

Dorit S Hochbaum. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations research*, 56(4):992–1009, 2008.

T Chiang Hu. Multi-commodity network flows. *Operations research*, 11(3):344–360, 1963.

Alon Itai. Two-commodity flow. *Journal of the ACM (JACM)*, 25(4):596–611, 1978.

Anil Kamath and Omri Palmon. Improved interior point algorithms for exact and approximate solution of multicommodity flow problems. In *SODA*, volume 95, pages 502–511. Citeseer, 1995.

Sanjiv Kapoor and Pravin M Vaidya. Speeding up karmarkar's algorithm for multicommodity flows. *Mathematical programming*, 73(1):111–127, 1996.

Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014.

Jeff L Kennington. Solving multicommodity transportation problems using a primal partitioning simplex technique. *Naval Research Logistics Quarterly*, 24(2):309–325, 1977.

Torbjörn Larsson and Di Yuan. An augmented lagrangian algorithm for large scale multicommodity routing. *Computational Optimization and Applications*, 27(2):187–215, 2004.

Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)*, 46(6):787–832, 1999.

Tom Leighton, Fillia Makedon, Serge Plotkin, Clifford Stein, Eva Tardos, and Spyros Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.

Pengfei Liu. A combinatorial algorithm for the multi-commodity flow problem. *arXiv preprint arXiv:1904.09397*, 2019.

Aleksander Madry. Computing maximum flow with augmenting electrical flows. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 593–602. IEEE, 2016.

Leonardo FS Moura, Luciano P Gaspary, and Luciana S Buriol. A branch-and-price algorithm for the single-path virtual network embedding problem. *Networks*, 71(3):188–208, 2018.

Adamou Ouorou, Philippe Mahey, and J-Ph Vial. A survey of algorithms for convex multicommodity flow problems. *Management science*, 46(1):126–147, 2000.

Robert Robey and Yuliana Zamora. *Parallel and high performance computing*. Simon and Schuster, 2021.

Khodakaram Salimifard and Sara Bigharaz. The multicommodity network flow problem: state of the art classification, applications, and solution methods. *Operational Research*, 22(1):1–47, 2022.

Rina R Schneur and James B Orlin. A scaling algorithm for multicommodity flow problems. *Operations Research*, 46 (2):231–246, 1998.

Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2): 318–334, 1990.

Bala Shetty and R Muthukrishnan. A parallel projection for the multicommodity network model. *Journal of the Operational Research Society*, 41(9):837–842, 1990.

Daniel A Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

Jan Van Den Brand, Li Chen, Richard Peng, Rasmus Kyng, Yang P Liu, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023.

I-Lin Wang. Multicommodity network flows: A survey, part i: Applications and formulations. *International Journal of Operations Research*, 15(4):145–153, 2018a.

I-Lin Wang. Multicommodity network flows: A survey, part ii: Solution methods. *International Journal of Operations Research*, 15(4):155–173, 2018b.

Yue Yu, Dan Calderone, Sarah HQ Li, Lillian J Ratliff, and Behçet Açıkmeşe. Variable demand and multi-commodity flow in markovian network equilibrium. *Automatica*, 140:110224, 2022.

Stavros A Zenios, Mustafa C Pinar, and Ron S Dembo. A smooth penalty function algorithm for network-structured problems. *European Journal of Operational Research*, 83(1):220–236, 1995.

Fangzhao Zhang and Stephen Boyd. Solving large multicommodity network flow problems on gpus. *arXiv preprint arXiv:2501.17996*, 2025.