Solving Large Multicommodity Network Flow Problems on GPUs

Fangzhao Zhang Stephen Boyd

January 26, 2025

Abstract

We consider the all-pairs multicommodity network flow problem on a network with capacitated edges. The usual treatment keeps track of a separate flow for each source-destination pair on each edge; we rely on a more efficient formulation in which flows with the same destination are aggregated, reducing the number of variables by a factor equal to the size of the network. Problems with hundreds of nodes, with a total number of variables on the order of a million, can be solved using standard generic interior-point methods on CPUs; we focus on GPU-compatible algorithms that can solve such problems much faster, and in addition scale to much larger problems, with up to a billion variables. Our method relies on the primal-dual hybrid gradient algorithm, and exploits several specific features of the problem for efficient GPU computation. Numerical experiments show that our primal-dual multicommodity network flow method accelerates state of the art generic commercial solvers by $100\times$ to $1000\times$, and scales to problems that are much larger. We provide an open source implementation of our method.

1 Multicommodity network flow optimization

1.1 Multicommodity network flow problem

Our formulation of the multicommodity network flow (MCF) problem, given below, follows [YDLB19].

Network. We consider a directed network with n nodes and m edges which is completely connected, *i.e.*, there is a directed path between each pair of nodes. Let $A \in \mathbf{R}^{n\times m}$ denote its incidence matrix, *i.e.*,

$$A_{i\ell} = \begin{cases} +1 & \text{edge } \ell \text{ enters node } i \\ -1 & \text{edge } \ell \text{ leaves node } i \\ 0 & \text{otherwise.} \end{cases}$$

Edge ℓ has a positive capacity c_ℓ. The total flow on edge ℓ (to be defined below) cannot exceed c_ℓ.

Traffic matrix. We consider the all-pairs multicommodity flow setting, *i.e.*, there is traffic that originates at every node, destined for every other node. We characterize the traffic between all source-destination pairs via the traffic matrix $T \in \mathbf{R}^{n\times n}$. For any pair of distinct nodes i, j, $T_{ij} \geq 0$ is the traffic from (source) node j to (destination) node i. There is no traffic from a node to itself; for mathematical convenience we define the diagonal traffic matrix entries as $T_{ii} = -\sum_{j\neq i} T_{ij}$, the negative of the total traffic with destination node i. With this definition of the diagonal entries we have $T\mathbf{1} = 0$, where $\mathbf{1}$ is the vector with all entries one.

Network utility. Let u_{ij} denote the strictly concave increasing utility function for traffic from node j to node i, for $j \neq i$. We will assume utility functions are differentiable with domains $\mathbf{R}_{++}$, the set of positive numbers. (The methods we describe are readily extended to nondifferentiable utilities using subgradients instead of gradients.) The total utility, which we wish to maximize, is $\sum_{i\neq j} u_{ij}(T_{ij})$. For simplicity we take $u_{ii} = 0$, so we can write the total utility as

$$U(T) = \sum_{i,j} u_{ij}(T_{ij}).$$

The domain of U is $\mathcal{T} = \{T \mid T_{ij} > 0 \text{ for } i \neq j\}$, *i.e.*, the traffic matrix must have positive off-diagonal entries.

Common examples of utility functions include the weighted log utility $u(s) = w \log s$, and the weighted power utility $u(s) = ws^\gamma$, with $\gamma \in (0, 1)$, where $w > 0$ is the weight.

Destination-based flow matrix. Following [YDLB19] we aggregate all flows with the same destination, considering it to be one commodity that is conserved at all nodes except the source and destination, but can be split and combined. The commodity flows are given

by the (destination-based) flow matrix $F \in \mathbf{R}^{n\times m}$, where $F_{i\ell} \geq 0$ denotes the flow on edge ℓ that is destined to node i. The edge capacity constraint can be expressed as $F^T\mathbf{1} \leq c$, where the inequality is elementwise.

Flow conservation. The flow destined for node i is conserved at all nodes $j \neq i$, including the additional injection of traffic T_{ij} that originates at node j and is destined for node i. This means that

$$T_{ij} + \sum_{\ell} A_{j\ell}F_{i\ell} = 0, \quad i, j = 1, \ldots, n, \quad j \neq i.$$

At the destination node, all traffic exits and we have (using our definition of T_{ii})

$$T_{ii} + \sum_{\ell} A_{i\ell}F_{i\ell} = 0, \quad i = 1, \ldots, n.$$

Combining these two, and using our specific definition of T_{ii}, flow conservation can be compactly written in matrix notation as

$$T + FA^T = 0.$$

Multicommodity flow problem. In the MCF problem we seek a flow matrix that maximizes total network utility, subject to the edge capacity and flow conservation constraints. This can be expressed as the problem

$$\begin{array}{ll} \text{maximize} & U(T) \\ \text{subject to} & F \geq 0, \quad F^T\mathbf{1} \leq c, \quad T + FA^T = 0, \end{array} \tag{1}$$

with variables F and T, and implicit constraint $T \in \mathcal{T}$. The problem data are the network topology A, edge capacities c, and the traffic utility functions u_{ij}.

We can eliminate the traffic matrix T using $T = -FA^T$ and state the MCF problem in terms of the variable F alone as

$$\begin{array}{ll} \text{maximize} & U(-FA^T) \\ \text{subject to} & F \geq 0, \quad F^T\mathbf{1} \leq c, \end{array} \tag{2}$$

with variable F, and implicit constraint $-FA^T \in \mathcal{T}$. The number of scalar variables in this problem is nm. For future use we define the feasible flow set as

$$\mathcal{F} = \{F \mid F \geq 0,\ F^T\mathbf{1} \leq c\}.$$

Existence and uniqueness of solution. The MCF problem (1) always has solution. To see that it is always feasible, consider a unit flow from each source to each destination, over the shortest path, *i.e.*, smallest number of edges, which exists since the graph is completely connected. We denote this flow matrix as F^{sp}. Now take $F = \alpha F^{\text{sp}}$, where $\alpha = 1/\max_i((F^{\text{sp}T}\mathbf{1})_i/c_i)$, so we have $F^T\mathbf{1} \leq c$. Evidently F is feasible, and we have $T_{ij} = \alpha$ for $i \neq j$. Existence of a solution follows since feasible set is bounded. The solution need not be unique. The optimal T, however, is unique.

Solving MCF. The multicommodity flow problem (2) is convex [BV04], and so in principal can be efficiently solved. In [YDLB19] the authors use standard generic interior-point solvers such as the commercial solver MOSEK [ApS19], together with CVXPY [DB16], to solve instances of the problem with tens of nodes, and thousands of variables, in a few seconds on a CPU. In this paper we introduce an algorithm for solving the MCF problem that fully exploits GPUs. For small and medium size problems our method gives a substantial speedup over generic methods; in addition it scales to much larger problems that cannot be solved by generic methods.

1.2 Optimality condition and residual

Optimality condition. A flow matrix F is optimal for (2) if and only if $F \in \mathcal{F}$, $-FA^T \in \mathcal{T}$, and for all $Z \in \mathcal{F}$,

$$\mathbf{Tr}(Z - F)^T G \geq 0,$$

where $G = \nabla_F(-U)(-FA^T)$ (see, *e.g.*, [BV04, §4.2.3]). We have $G = U'A$, where $U'_{ij} = u'_{ij}((-FA^T)_{ij})$.

We can express this optimality condition in terms of projection onto the feasible flow set. Let Π denote Euclidean projection onto $\mathcal{F}$. The optimality condition can be expressed in terms of a matrix $Q \in \mathbf{R}^{n\times m}$, with $F = \Pi(Q)$. Evidently $F \in \mathcal{F}$. We need to have $-FA^T \in \mathcal{T}$ for the gradient G to exist. When that holds, the other optimality condition can be expressed as

$$G = \gamma(F - Q)$$

for some $\gamma \geq 0$.

Optimality residual. For any $Q \in \mathbf{R}^{n\times m}$ with $F = \Pi(Q)$, we define the (optimality) residual as

$$r(Q) = \begin{cases} \min_{\gamma\geq 0} \|G - \gamma(F - Q)\|_F^2 & -FA^T \in \mathcal{T} \\ \infty & \text{otherwise,} \end{cases}$$

where $\|\cdot\|_F^2$ denotes the squared Frobenius norm of a matrix, *i.e.*, the sum of squares of its entries. When $-FA^T \in \mathcal{T}$, the righthand side is a quadratic function of γ, so the minimum is easily expressed explicitly as

$$r(Q) = \begin{cases} \|G\|_F^2 - \frac{\mathbf{Tr}^2\, G^T(F-Q)}{\|F-Q\|_F^2} & -FA^T \in \mathcal{T},\ F \neq Q,\ \mathbf{Tr}\, G^T(F - Q) \geq 0 \\ \|G\|_F^2 & -FA^T \in \mathcal{T},\ F = Q \text{ or } \mathbf{Tr}\, G^T(F - Q) < 0 \\ \infty & \text{otherwise.} \end{cases} \tag{3}$$

Evidently $F = \Pi(Q)$ is optimal if and only if $r(Q) = 0$.

1.3 Related work

Multicommodity network flow. Historically, different forms of MCF problems have been formulated and studied. Starting from [FJF58] and [Hu63] which studied a version

with linear utility functions, which can be formulated as a linear program, later works develop nonlinear convex program formulations [GG95, OM00] and (nonconvex) mixed integer program formulations [Man12, KS16, Zan05] of MCF problems for different application purposes. These various forms of MCF have been widely used in transportation management [EMS05, MMPP15, RFZS16], energy and economic sectors [Sin78, GG95, Man12], and network communication [WRP+06, KS16, LHB17]. [SB22] surveys over two hundred studies on MCF problems between 2000 and 2019. In this work, we focus on nonlinear convex formulation of MCF problems and develop GPU-compatible algorithms for solving large problem instances. See [OMV00] for a survey on nonlinear convex MCF problems. MCF models have very recently been exploited to design multi-GPU communication schedules for deep learning tasks [LAK+24, BZF+24], but the underlying MCF problems are solved with CPU-based solvers.

First-order methods for convex optimization. First-order methods such as gradient descent algorithm, proximal point algorithm, primal-dual hybrid gradient algorithm, and their accelerated versions have been exploited to tackle different forms of convex optimization problems. Compared to second-order methods which exploit Hessian information, first-order methods are known for their low computational complexity and are thus attractive for solving large-scale optimization problems. Recently, primal-dual hybrid gradient algorithm has been explored for solving large linear programs [ADH+21, LY24, LPY24] and optimal transport problems [RCLO18] on GPUs. Other first-order methods such as ADMM have been exploited for designing GPU-accelerated optimizers for optimal power flow problems [DGR24, RBK25].

GPU-accelerated network flow optimization. Specialized to GPU-based optimizers for network flow optimization, [WZR+18] considers implementing a parallel routing algorithm on GPUs for SDN networks, which solves the Lagrangian relaxation of a mixed integer linear program. [KOY+15] implements a genetic method on GPUs for solving an integer linear program formulation of routing problem. [ZAC23] considers a linear program formulation of multicommodity network flow problems and constructs a deep learning model for generating new columns in delayed column generation method. [WHH12] implements an asynchronous push-relabel algorithm for single commodity maximum network flow problem, which is CPU-GPU hybrid. [LHQ+24] exploits exactly the same flow aggregation formulation of MCF following [YDLB19] as we do and trains a neural network model for minimizing unconstrained Lagrangian relaxation objective, and feeds the result as warm start to Gurobi [Gur24] to get the final answer.

1.4 Contribution

Motivated by recent advancement of GPU optimizers, in this work we seek to accelerate large-scale nonlinear convex MCF problem solving with GPUs. Specifically, we adopt the MCF problem formulation in [YDLB19] (also described above) which is compactly matrix-represented and requires fewer optimization variables by exploiting flow aggregation. We

show that this specific problem formulation can be efficiently solved with first-order primal-dual hybrid gradient method when run on GPUs.

To the best of our knowledge, our work is the first to tackle exactly solving convex MCF problems on GPUs. Classic works for solving such large-scale MCF problems usually adopt Lagrangian relaxation for the coupling constraint and solve the resulting subproblems with smaller sizes in parallel (see, *e.g.*, [OMV00]). In our work, we do not exploit any explicit problem decomposition strategy and our algorithmic acceleration is mainly empirical and depends on highly-optimized CUDA kernels for matrix operations. Moreover, we achieve problem size reduction via flow aggregation. Therefore our method has a simpler form which does not involve massive subproblem solving and synchronizing, and is also exact.

1.5 Outline

We describe our algorithm in §2. Experimental results, using our PyTorch implementation, are presented and discussed in §3; very similar results obtained with our JAX implementation are given in appendix B. We conclude our work in §4. The code, and all data needed to reproduce the results reported in this paper, can be accessed at

`https://github.com/cvxgrp/pdmcf`.

2 Primal-dual hybrid gradient

2.1 Primal-dual saddle point formulation

We first derive a primal-dual saddle point formulation of the MCF problem (1). Let $\mathcal{I}$ denote the indicator function of $\mathcal{F}$, *i.e.*, $\mathcal{I}(F) = 0$ for $F \in \mathcal{F}$ and $\mathcal{I}(F) = \infty$ otherwise. We switch to minimizing $-U$ in (1) to obtain the equivalent problem

$$\begin{array}{ll} \text{minimize} & -U(T) + \mathcal{I}(F) \\ \text{subject to} & T = -FA^T, \end{array} \tag{4}$$

with variables T and F. We introduce a dual variable $Y \in \mathbf{R}^{n\times n}$ associated with the (matrix) equality constraint. The Lagrangian is then

$$\mathcal{L}(T, F; Y) = -U(T) + \mathcal{I}(F) - \mathbf{Tr}\, Y^T(T + FA^T)$$

(see [BV04, Chap. 5]). The Lagrangian $\mathcal{L}$ is convex in the primal variables (T, F) and affine (and therefore concave) in the dual variable Y. If $(T, F; Y)$ is a saddle point of $\mathcal{L}$, then (T, F) is a solution to problem (4) (and F is a solution to the MCF problem (2)); the converse also holds.

We can analytically minimize $\mathcal{L}$ over T to obtain the reduced Lagrangian

$$\hat{\mathcal{L}}(F; Y) = \inf_T \mathcal{L}(T, F; Y) = -(-U)^*(Y) + \mathcal{I}(F) - \mathbf{Tr}\, Y^T FA^T, \tag{5}$$

where U^* is the conjugate function of U [BV04, §3.3]. This reduced Lagrangian is convex in the primal variable F and concave in the dual variable Y. If $(F;Y)$ is a saddle point of $\hat{\mathcal{L}}$, then F is a solution to the MCF problem (2) (see [MP18, § 1]). We observe that $\hat{\mathcal{L}}$ is convex-concave, with a bilinear coupling term.

2.2 Basic PDHG method

The primal-dual hybrid gradient (PDHG) algorithm, as first introduced in [ZC08] and later studied in [CP11, CP15], is a first-order method for finding a saddle point of a convex-concave function with bilinear coupling term. The algorithm was extended to include over-relaxation in [CP15, §4.1], which has been observed to improve convergence in practice. For (5), PDHG has the form

$$\begin{array}{lcl} \hat{F}^{k+1/2} & = & \mathbf{prox}_{\alpha\mathcal{I}}(F^{k-1/2} + \alpha Y^k A) \\ F^{k+1} & = & 2\hat{F}^{k+1/2} - F^{k-1/2} \\ \hat{Y}^{k+1} & = & \mathbf{prox}_{\beta(-U)^*}(Y^k - \beta F^{k+1} A^T) \\ F^{k+1/2} & = & \rho\hat{F}^{k+1/2} + (1-\rho)F^{k-1/2} \\ Y^{k+1} & = & \rho\hat{Y}^{k+1} + (1-\rho)Y^k \end{array} \tag{6}$$

where $\mathbf{prox}_f(v) = \mathrm{argmin}_x(f(x) + (1/2)\|x - v\|_2^2)$ denotes the proximal operator of f [PB14], $\alpha, \beta > 0$ are positive step sizes satisfying $\alpha\beta \leq 1/\|A\|_2^2$, and $\rho \in (0, 2)$ is the over-relaxation parameter.

Reasonable choices for the parameters are

$$\alpha = \beta = 1/\|A\|_2, \qquad \rho = 1.9.$$

(An upper bound on $\|A\|_2$ can be used in place of $\|A\|_2$.)

Convergence. In [CP15] it has been shown that when there exists a saddle point of $\hat{\mathcal{L}}$, $(F^k; Y^k)$ converges to a saddle point of $\hat{\mathcal{L}}$ as $k \to \infty$. For MCF the existence of an optimal flow matrix and dual variable is known, so F^k converges to an optimal flow matrix. It follows that $r(F^{k-1/2} + \alpha Y^k A) \to 0$ as $k \to \infty$. We note that $-FA^T \in \mathcal{T}$ only holds eventually.

2.3 Proximal operators

Here we take a closer look at the two proximal operators appearing in PDHG.

First proximal operator. We note that $\mathbf{prox}_{\alpha\mathcal{I}}$ appearing in the $\hat{F}^{k+1/2}$ update of (6) is projection onto $\mathcal{F}$,

$$\mathbf{prox}_{\alpha\mathcal{I}}(F) = \Pi(F).$$

Since the constraints that define $\mathcal{F}$ separate across the columns of F, we can compute $\Pi(F)$ by projecting each column f_ℓ of F onto the scaled simplex $\mathcal{S}_\ell = \{f \mid f \geq 0,\ \mathbf{1}^T f \leq c_\ell\}$. This projection has the form

$$\Pi_{\mathcal{S}_\ell}(f_\ell) = (f_\ell - \mu_\ell \mathbf{1})_+,$$

where μ_ℓ is the optimal Lagrange multiplier and $(a)_+ = \max\{a, 0\}$, which is applied elementwise to a vector. The optimal μ_ℓ is the smallest nonnegative value for which $(f_\ell - \mu_\ell \mathbf{1})_+^T \mathbf{1} \leq c_\ell$. This is readily found by a bisection algorithm; see §2.6.

Second proximal operator. The proximal operator appearing in the $\hat{Y}^{k+1}$ update step in (6) can be decomposed entrywise, since $\beta(-U)^*$ is a sum of functions of different variables. (The diagonal entries $-u_{ii}$ are zero, so $(-\beta u_{ii})^*$ is the indicator function of $\{0\}$, and its proximal operator is the zero function.) For each off-diagonal entry $i \neq j$ we need to evaluate

$$\mathbf{prox}_{\beta(-u_{ij})^*}(y).$$

These one-dimensional proximal operators are readily computed in the general case. For the weighted log utility $u(s) = w \log s$, we have

$$\mathbf{prox}_{\beta(-u)^*}(y) = \frac{y - \sqrt{y^2 + 4\beta w}}{2}.$$

For the weighted power utility $u(s) = ws^\gamma$, $\mathbf{prox}_{\beta(-u)^*}(y)$ is the unique negative number z for which

$$(-z)^{c_1+2} + y(-z)^{c_1+1} - c_1 c_2 = 0,$$

where

$$c_1 = \frac{\gamma}{1-\gamma} > 0, \quad c_2 = \beta \left(\frac{1}{\gamma} - 1 \right) (w\gamma)^{\frac{1}{1-\gamma}} > 0.$$

2.4 Adaptive step sizes

In the basic PDHG algorithm (6), the step sizes α and β are fixed. It has been observed that varying them adaptively as the algorithm runs can improve practical convergence substantially [ADH+21]. We describe our implementation of adaptive step sizes here.

We express the step sizes as

$$\alpha^k = \eta/\omega^k, \qquad \beta^k = \eta\omega^k,$$

where $\eta \leq 1/\|A\|_2$ and $\omega^k > 0$ gives the primal weight. With $\omega^k = 1$ we obtain basic PDHG (6).

The primal weight ω^k is initialized as $\omega^0 = 1$ and adapted following [ADH+21, §3.3] as

$$\omega^{k+1} = \left(\frac{\Delta_Y^{k+1}}{\Delta_F^{k+1}} \right)^\theta \left(\omega^k \right)^{1-\theta}, \tag{7}$$

where $\Delta_F^{k+1} = \|F^{k+1/2} - F^{k-1/2}\|_F$, $\Delta_Y^{k+1} = \|Y^{k+1} - Y^k\|_F$ and θ is a parameter fixed as 0.5 in our implementation. The intuition behind the primal weight update (7) is to balance the primal and dual residuals; see [ADH+21, §3.3] for details. In [ADH+21] the authors update ω each restart. We do not use restarts, and have found that updating ω^k every k^{adapt}

iterations, when both $\Delta_F^k > 10^{-5}$ and $\Delta_Y^k > 10^{-5}$ hold, works well in practice for MCF. In our experiments we use $k^{\text{adapt}} = 100$. We can also stop adapting ω^k after some number of iterations, keeping it constant in future iterations. At least technically this implies that the convergence proof for constant ω holds for the adaptive algorithm.

A simple bound on $\|A\|_2$. We can readily compute a simple upper bound on

$$\|A\|_2 = \sqrt{\lambda_{\max}(AA^T)},$$

where $\lambda_{\max}$ denotes the maximum eigenvalue. We observe that AA^T is the Laplacian matrix associated with the network, for which the well-known bound

$$\lambda_{\max}(AA^T) \leq 2d_{\max}$$

holds, where $d_{\max}$ is the largest node degree in the graph. (For completeness we derive this in appendix A.) Thus we can take

$$\eta = 1/\sqrt{2d_{\max}}. \tag{8}$$

2.5 Algorithm

We summarize our final algorithm, which we call PDMCF. We set $r^0 = +\infty$, $\alpha^0 = \eta/\omega^0$, and $\beta^0 = \eta\omega^0$, where η is given in (8) and $\omega^0 = 1$.

Algorithm 2.1 PDMCF

given $F^{-1/2}, Y^0$, parameter $\epsilon > 0$.

for $k = 0, 1, \ldots$

1. *Check stopping criterion.* Quit and return $\hat{F}^{k-1/2}$ if $r^k < nm\epsilon$ holds.
2. *Basic PDHG updates* (6).
   $\hat{F}^{k+1/2} = \Pi(F^{k-1/2} + \alpha^k Y^k A)$.
   $F^{k+1} = 2\hat{F}^{k+1/2} - F^{k-1/2}$.
   $\hat{Y}_{ij}^{k+1} = \begin{cases} \mathbf{prox}_{\beta^k(-u_{ij})^*}(Y_{ij}^k - \beta^k(F^{k+1}A^T)_{ij}) & j \neq i \\ 0 & j = i. \end{cases}$
   $F^{k+1/2} = \rho\hat{F}^{k+1/2} + (1-\rho)F^{k-1/2}$.
   $Y^{k+1} = \rho\hat{Y}^{k+1} + (1-\rho)Y^k$.
3. *Adaptive step size updates* (7) (if k is multiple of k^{adapt} and $\Delta_F^{k+1}, \Delta_Y^{k+1} > \tau$).
   $\omega^{k+1} = \left(\Delta_Y^{k+1}/\Delta_F^{k+1}\right)^\theta \left(\omega^k\right)^{1-\theta}$.
   $\alpha^{k+1} = \eta/\omega^{k+1}, \qquad \beta^{k+1} = \eta\omega^{k+1}$.

Initialization. We always take $F^{-1/2} = 0$ and $Y^0 = I - \mathbf{1}\mathbf{1}^T$. We can alternatively use a better guess of $F^{-1/2}$ and Y, for example in a warm start, when we have already solved a problem with similar data. We illustrate more on this in §3.1.

Stopping criterion. Since $\hat{F}^{k+1/2}$ is result of projection onto $\mathcal{F}$, our optimality residual (3) has the form

$$r^{k+1} = r(F^{k-1/2} + \alpha^k Y^k A).$$

We consider the stopping criterion $r^k < nm\epsilon$, *i.e.*, the entrywise normalized residual r^k/nm is smaller than a user-specified threshold ϵ.

2.6 Implementation details

Incidence matrix indexing. We only store the indices of the non-zero entries of A. Matrix multiplication with A and A^T can be efficiently computed by exploiting scatter and gather functions, which are highly optimized CUDA kernels and are available in most major GPU computing languages.

Projection onto scaled simplex. To compute μ_ℓ when $(f_\ell)_+^T \mathbf{1} > c_\ell$, we follow [HWC74] and first sort f_ℓ from largest entry to smallest entry to form f'_ℓ. We then find the largest index t such that $f'_{\ell t} - ((\sum_{i=1}^t f'_{\ell i} - c_\ell)/t) > 0$. Finally we take $\mu_\ell = (\sum_{i=1}^t f'_{\ell i} - c_\ell)/t$.

3 Experiments

We run all our experiments on a single H100 GPU with 80 Gb of memory supported by 26 virtual CPU cores and 241 Gb of RAM. The results given below are for our PyTorch implementation; similar results, reported in appendix B, are obtained with our JAX implementation.

3.1 Examples

Data and parameters. We consider weighted log utilities of form $u_{ij}(T_{ij}) = w_{ij} \log T_{ij}$. We take $\log w_{ij}$ to be uniform on $[\log 0.3, \log 3]$. For network topology, we first create n two-dimensional data points $\xi_i \in \mathbf{R}^2$, each denoted by (ξ_{ix}, ξ_{iy}) for $i = 1, \ldots, n$. We take ξ_{ix} and ξ_{iy} uniform on $[0, 1]$. Then we add both edges (ξ_i, ξ_j) and (ξ_j, ξ_i) when either ξ_i is among the q-nearest neighbors of ξ_j or vice versa. For each edge ℓ, we impose edge capacity c_ℓ where we take $\log c_\ell$ to be uniform on $[\log 0.5, \log 5]$.

We use stopping criterion threshold $\epsilon = 0.01/(n(n-1))$ for small to medium size problems and $\epsilon = 0.03/(n(n-1))$ for large size problems. We compare to CPU-based commercial solver MOSEK, with default settings. MOSEK is able to solve the problems to high accuracy; we have checked that for all problem instances, the normalized utility differences between results of PDMCF and MOSEK are no more than around 0.01. The pairwise normalized (optimal) utilities range between around 1 and 10, which means that PDMCF finds flows that are between 0.1% and 1% suboptimal compared to the flows found by MOSEK.

problem sizes				timing (s)			iterations
n	q	m	nm	MOSEK	PDMCF (CPU)	PDMCF (GPU)	
100	10	1178	1×10^5	5	1	0.5	490
200	10	2316	5×10^5	23	2	0.7	690
300	10	3472	1×10^6	95	6	0.8	840
500	10	5738	3×10^6	340	18	1.1	950
500	20	11176	6×10^6	1977	34	1.4	790
1000	10	11424	1×10^7	2889	1382	19.5	7220
1000	20	22286	2×10^7	16765	349	5.1	1040

Table 1: Runtime table for small and medium size problems.

problem sizes				timing (s)			iterations
n	q	m	nm	MOSEK	PDMCF (CPU)	PDMCF (GPU)	
3000	10	34424	1×10^8	OOM	7056	96	4140
5000	10	57338	3×10^8	OOM	19152	395	3970
10000	10	114054	1×10^9	OOM	87490	1908	4380

Table 2: Runtime table for large size problems.

Small to medium size problems. Table 1 shows runtime for both MOSEK and PDMCF required to solve problem instances of various sizes. The column titled nm gives the number of scalar optimization variables in the problem instance. We see that our implementation of PDMCF on a GPU gives a speedup over MOSEK of 10× to 1000×, with more significant speedup for larger problem instances. We also report runtime for PDMCF when run on CPU, which is still quicker than MOSEK but with a significantly lower speedup. Similar performance is also observed for our JAX implementation, reported in appendix B.

Large size problems. Table 2 shows runtime for several large problem instances. MOSEK fails to solve all these problems due to memory limitations. PDMCF handles all these problem instances, with the largest one involving 10^9 variables.

Scaling. We scatter plot the runtime data for small and medium problem instances in figure 1. Here we take 5 problem instances generated by iterating over random seeds $\{0, 1, 2, 3, 4\}$ for the different n, q values listed in table 1. The x-axis represents optimization variable size nm and the y-axis represents runtime in seconds. We plot on a log-log scale. The lines show the affine function fits to these data, with a slope around 1.5 for MOSEK and around 0.5 for PDMCF.

Convergence plot. Figure 2 shows the convergence for three problem instances with variable sizes $10^5, 10^6$, and 10^7 with PDMCF, where the x-axis represents iteration numbers. Especially in the initial iterations we have infinite residual r^k since $-FA^T \notin \mathcal{T}$. For those

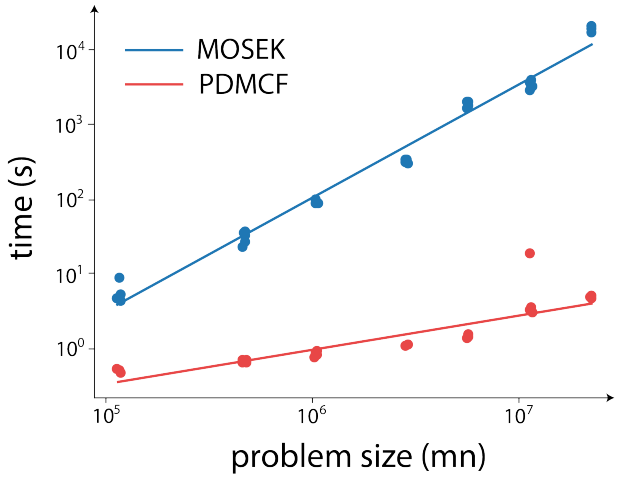

Figure 1: Runtime plot for small and medium size problems.

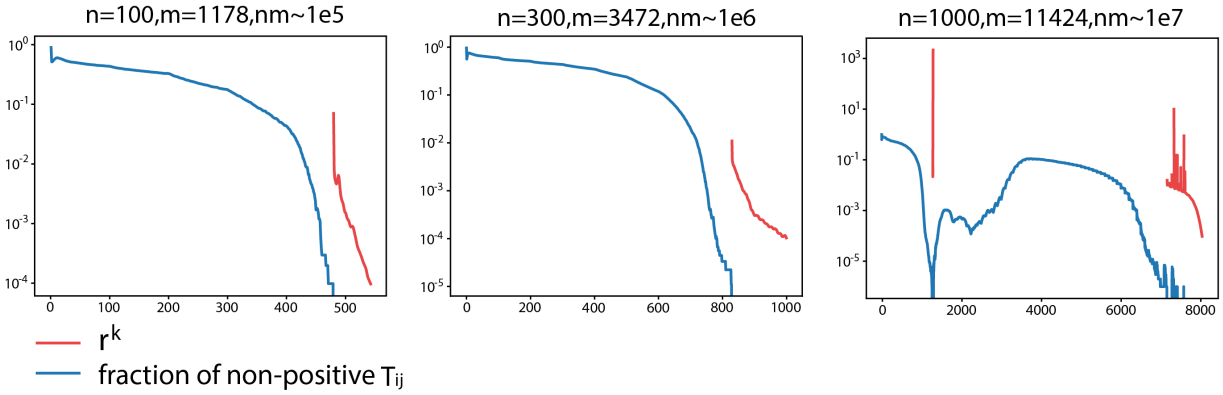

Figure 2: Convergence plot for small and medium size problems.

iterations we plot the fraction of nonpositive off-diagonal entries of T in blue. For feasible iterates we plot the (finite) residual, in red.

Warm start. In §2.5 we start with some simple initial $F^{-1/2}$ and Y^0. We also test performance of PDMCF with warm starts. In figure 3 we present how runtime changes under different warm starts. To form these warm starts, for some perturbation ratio ν, we randomly perturb entries of our utility weight matrix to derive $\tilde{w}_{ij} = (1 \pm \nu)w_{ij}$, each with probability a half. We solve the multicommodity network flow problem with perturbed utility weight $\tilde{w}$ with PDMCF until we land at a feasible point $(F^{\text{feas}}, Y^{\text{feas}})$ satisfying $(-F^{\text{feas}}A^T)_{ij} > 0$ for all distinct i, j. We record the primal weight at this point as ω^{feas}. We then solve the desired multicommodity network flow problem with original utility weight w with $F^{-1/2} = F^{\text{feas}}, Y^0 = Y^{\text{feas}}$ and $\omega^0 = \omega^{\text{feas}}$. We note that setting $\omega^0 = \omega^{\text{feas}}$ is impor-

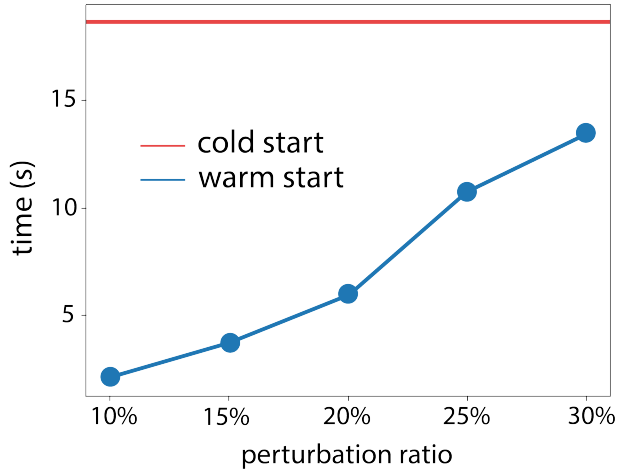

Figure 3: Warm start plot for medium size problem.

tant for accelerated convergence, otherwise it usually requires similar number of iterations to converge as cold start if we simply set $\omega^0 = 1$. In figure 3, we take problem instance with $n = 1000, q = 10$. x-axis stands for perturbation ratio ν and y-axis represents runtime in seconds. As can be observed, with perturbation ratio $\nu = 10\%$, we harness $> 80\%$ saving of runtime. Such savings keep decreasing to around 30% when $\nu = 30\%$, which makes sense given that larger perturbation indicates more different utility weights between original problem and perturbed problem, thus our warm start is expected to stay further from optimal solution to the original problem instance.

4 Conclusion

In this work, we present PDMCF algorithm which accelerates solving multicommodity network flow problems on GPUs. Our method starts with a destination-based formulation of multicommodity network flow problems which reduces optimization variable amount compared to classic problem formulation. We then apply PDHG algorithm to solve this destination-based problem formulation. Empirical results verify that our algorithm is GPU-friendly and brings up to three orders of magnitude of runtime acceleration compared to classic CPU-based commercial solvers. Moreover, our algorithm is able to solve ten times larger problems than those can be solved by commercial CPU-based solvers.

Acknowledgements

We thank Anthony Degleris and Parth Nobel for valuable discussions on implementation details.

References

[ADH+21] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. In *Neural Information Processing Systems*, 2021.

[ApS19] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[BZF+24] Prithwish Basu, Liangyu Zhao, Jason Fantl, Siddharth Pal, Arvind Krishnamurthy, and Joud Khoury. Efficient all-to-all collective communication schedules for direct-connect topologies. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '24, page 28–41. Association for Computing Machinery, 2024.

[CP11] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40:120–145, 2011.

[CP15] Antonin Chambolle and Thomas Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159:253–287, 2015.

[DB16] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[DGR24] Anthony Degleris, Abbas El Gamal, and Ram Rajagopal. GPU accelerated security constrained optimal power flow. *arXiv preprint arXiv:2410.17203*, 2024.

[EMS05] Alan Erera, Juan Morales, and Martin Savelsbergh. Global intermodal tank container management for the chemical industry. *Transportation Research Part E: Logistics and Transportation Review*, 41:551–566, 2005.

[FJF58] Lester Randolph Ford Jr and Delbert Ray Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[GG95] Antoine Gautier and Frieda Granot. Forest management: A multicommodity flow formulation and sensitivity analysis. *Management Science*, 41(10):1654–1668, 1995.

[Gur24] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[Hu63] Te Chiang Hu. Multi-commodity network flows. *Operations Research*, 11(3):344–360, 1963.

[HWC74] Michael Held, Philip Wolfe, and Harlan Pinkney Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[KOY+15] Ko Kikuta, Eiji Oki, Naoaki Yamanaka, Nozomu Togawa, and Hidenori Nakazato. Effective parallel algorithm for GPGPU-accelerated explicit routing optimization. In *2015 IEEE Global Communications Conference*, pages 1–6. IEEE, 2015.

[KS16] Ozgur Kabadurmus and Alice Smith. Multi-commodity k-splittable survivable network design problems with relays. *Telecommunication Systems*, 62:123–133, 2016.

[LAK+24] Xuting Liu, Behnaz Arzani, Siva Kesava Reddy Kakarla, Liangyu Zhao, Vincent Liu, Miguel Castro, Srikanth Kandula, and Luke Marshall. Rethinking machine learning collective communication as a multi-commodity flow problem. In *Proceedings of the ACM SIGCOMM 2024 Conference*, page 16–37. Association for Computing Machinery, 2024.

[LHB17] Safa Bhar Layeb, Riheb Heni, and Ali Balma. Compact MILP models for the discrete cost multicommodity network design problem. In *2017 International Conference on Engineering & MIS*, pages 1–7. IEEE, 2017.

[LHQ+24] Haoyu Liu, Siyong Huang, Shaoxiang Qin, Tianyi Yang, Tianze Yang, Qiao Xiang, and Xue Liu. Keep your paths free: Toward scalable learning-based traffic engineering. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, page 189–191. Association for Computing Machinery, 2024.

[LPY24] Haihao Lu, Zedong Peng, and Jinwen Yang. MPAX: Mathematical programming in JAX. *arXiv preprint arXiv:2412.09734*, 2024.

[LY24] Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in julia. *arXiv preprint arXiv:2311.12180*, 2024.

[Man12] Massimiliano Manfren. Multi-commodity network flow models for dynamic energy management–mathematical formulation. *Energy Procedia*, 14:1380–1385, 2012.

[MMPP15] Marta Mesquita, Margarida Moz, Ana Paias, and Margarida Pato. A decompose-and-fix heuristic based on multi-commodity flow models for driver rostering with days-off pattern. *European Journal of Operational Research*, 17, September 2015.

[MP18] Yura Malitsky and Thomas Pock. A first-order primal-dual algorithm with linesearch. *SIAM Journal on Optimization*, 28(1):411–432, 2018.

[OM00] Adam Ouorou and Philippe Mahey. Minimum mean cycle cancelling method for nonlinear multicommodity flow problems. *European Journal of Operational Research*, 121:532–548, February 2000.

[OMV00] Adamou Ouorou, Philippe Mahey, and Jean-Philippe Vial. A survey of algorithms for convex multicommodity flow problems. *Management science*, 46(1):126–147, 2000.

[PB14] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.

[RBK25] Minseok Ryu, Geunyeong Byeon, and Kibaek Kim. A GPU-accelerated distributed algorithm for optimal power flow in distribution systems. *arXiv preprint arXiv:2501.08293*, 2025.

[RCLO18] Ernest K Ryu, Yongxin Chen, Wuchen Li, and Stanley Osher. Vector and matrix optimal mass transport: theory, algorithm, and applications. *SIAM Journal on Scientific Computing*, 40(5):A3675–A3698, 2018.

[RFZS16] Andreas Rudi, Magnus Frohling, Konrad Zimmer, and Frank Schultmann. Freight transportation planning considering carbon emissions and in-transit holding costs: a capacitated multi-commodity network flow model. *EURO Journal on Transportation and Logistics*, 5:123–160, 2016.

[SB22] 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.

[Sin78] Ishita Singh. A dynamic multi-commodity model of the agricultural sector : a regional application in brazil. *European Economic Review*, 11:155–179, 1978.

[WHH12] Jiadong Wu, Zhengyu He, and Bo Hong. Chapter 5 - efficient CUDA algorithms for the maximum network flow problem. In *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 55–66. Morgan Kaufmann, Boston, 2012.

[WRP+06] Daniel Wagner, Gunther Raidl, Ulrich Pferschy, Petra Mutzel, and Peter Bachhiesl. A multi-commodity flow approach for the design of the last mile in real-world fiber optic networks. In *Operation Research Proceedings*, 2006.

[WZR+18] Xiong Wang, Qian Zhang, Jing Ren, Shizhong Xu, Sheng Wang, and Shui Yu. Toward efficient parallel routing optimization for large-scale SDN networks using GPGPU. *Journal of Network and Computer Applications*, 113:1–13, 2018.

[YDLB19] Ping Yin, Steven Diamond, Bill Lin, and Stephen Boyd. Network optimization for unified packet and circuit switched networks. *arXiv preprint arXiv:1808.00586*, 2019.

[ZAC23] Shuai Zhang, Oluwaseun Ajayi, and Yu Cheng. A self-supervised learning approach for accelerating wireless network optimization. *IEEE Transactions on Vehicular Technology*, 72(6):8074–8087, 2023.

[Zan05] Abdulhakim Zantuti. The capacity and non-simultaneously multicommodity flow problem in wide area network and data flow management. In *Proceedings of the 18th International Conference on Systems Engineering*, page 76–80. IEEE Computer Society, 2005.

[ZC08] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *UCLA CAM Report*, 34(2), 2008.

A Upper bound on $\lambda_{\max}(AA^T)$

For a directed graph with incidence matrix A, $d_i = (AA^T)_{ii}$ is the degree of node i and for $i \neq j$, $-(AA^T)_{ij}$ is the number of edges connecting node i and node j, *i.e.*, 2 if both edges (i,j) and (j,i) exist. Note that $\lambda_{\max}(AA^T) = \max_{\|x\|_2=1} x^T(AA^T)x = \max_{x\neq 0} \frac{x^T(AA^T)x}{x^Tx}$. We have

$$\begin{aligned} x^T(AA^T)x &= \textstyle\sum_i (AA^T)_{ii}x_i^2 + \sum_{i\neq j}(AA^T)_{ij}x_ix_j \\ &= \textstyle\sum_i d_ix_i^2 + \sum_{i\neq j}(AA^T)_{ij}x_ix_j \\ &\leq \textstyle\sum_i d_ix_i^2 + \sum_{i\neq j}|(AA^T)_{ij}|(x_i^2/2 + x_j^2/2) \\ &= \textstyle\sum_i x_i^2(d_i + \sum_{j\neq i}|(AA^T)_{ij}|) \\ &= \textstyle\sum_i 2d_ix_i^2 \\ &\leq 2d_{\max}x^Tx. \end{aligned}$$

Therefore $\lambda_{\max}(AA^T) = \max_{x\neq 0} \frac{x^T(AA^T)x}{x^Tx} \leq 2d_{\max}$.

B JAX results

The results shown in §3.1 are for our PyTorch implementation. Here we provide the same results for our JAX implementation. Tables 3 and 4 show the runtimes on the same problem instances as reported in tables 1 and 2. We note that JAX's just-in-time (JIT) compilation adds runtime overhead for first-time function compilation and thus it does worse than its PyTorch counterpart on small size problems. The runtimes of these two versions are close for medium and large size problems, with JAX slightly slower.

problem sizes				timing (s)			iterations
n	q	m	nm	MOSEK	PDMCF (CPU)	PDMCF (GPU)	
100	10	1178	1×10^5	5	12	5	490
200	10	2316	5×10^5	23	57	6	690
300	10	3472	1×10^6	95	164	6	840
500	10	5738	3×10^6	340	548	7	950
500	20	11176	6×10^6	1977	890	8	790
1000	10	11424	1×10^7	2889	18554	26	7150
1000	20	22286	2×10^7	16765	5143	15	1040

Table 3: Runtime table for small and medium size problems (JAX).

problem sizes				timing (s)			iterations
n	q	m	nm	MOSEK	PDMCF (CPU)	PDMCF (GPU)	
3000	10	34424	1×10^8	OOM	106274	139	4140
5000	10	57338	3×10^8	OOM	382400	421	3970
10000	10	114054	1×10^9	OOM		2078	4380

Table 4: Runtime table for large size problems (JAX).