

© 2025 г. Ф. ЧЖАН (zfzhao@stanford.edu),  
С. БОЙД, д-р философии (boyd@stanford.edu)  
(Стэнфордский университет, Стэнфорд, США)

## РЕШЕНИЕ ЗАДАЧ О МНОГОТОВАРНЫХ СЕТЕВЫХ ПОТОКАХ БОЛЬШОЙ РАЗМЕРНОСТИ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

Рассматривается задача о многотоварных сетевых потоках из всех пар узлов в сети с ребрами, имеющими заданные пропускные способности. При обычном подходе для каждой пары узлов “источник–назначение” на каждом ребре отслеживается отдельный поток. В статье используется более эффективная формулировка, в которой потоки с одним и тем же узлом-назначением объединяются, что позволяет уменьшить количество переменных в  $k$  раз, где  $k$  – размер сети. Задачи с сотнями узлов, с общим числом переменных порядка миллиона, могут быть решены стандартными общими методами внутренней точки на центральных процессорах (CPU); ниже используются совместимые с графическими процессорами (GPU) алгоритмы, которые могут решать такие задачи гораздо быстрее и, кроме того, масштабируются на гораздо большие задачи, с миллиардом переменных. Представленный метод основан на прямо-двойственном гибридном градиентном алгоритме и использует несколько особенностей задачи для эффективных вычислений на GPU. С помощью численных экспериментов показано, что прямо-двойственный метод многотоварных сетевых потоков ускоряет современные коммерческие решатели от 100 до 1000 раз и масштабируется на задачи гораздо большего размера. Приведена реализация данного метода с открытым исходным кодом.

*Ключевые слова:* многотоварные потоки, прямо-двойственный метод, GPU-совместимая оптимизация.

**DOI:** 10.31857/S0005231025080066, **EDN:** UTHPNJ

### 1. Введение

#### 1.1. Оптимизация многотоварных сетевых потоков

Формулировка задачи о многотоварных сетевых потоках (multicommodity network flow, MCF), приведенная ниже, следует [1].

**Сеть.** Рассмотрим направленную сеть с  $n$  узлами и  $m$  ребрами, которая является полносвязной, т.е. между каждой парой узлов существует направленный путь. Обозначим через  $A \in \mathbf{R}^{n \times m}$  ее матрицу инцидентности:

$$A_{i\ell} = \begin{cases} +1, & \text{если ребро } \ell \text{ входит в узел } i, \\ -1, & \text{если ребро } \ell \text{ выходит из узла } i, \\ 0 & \text{в остальных случаях.} \end{cases}$$

Ребро  $\ell$  имеет положительную пропускную способность  $c_\ell$ . Суммарный поток на ребре  $\ell$  (определен ниже) не может превышать  $c_\ell$ .

**Матрица трафика.** Рассмотрим постановку задачи с многотоварными потоками из всех пар, т.е. существует трафик, который исходит из каждого узла и предназначен для каждого другого узла. Будем характеризовать трафик между всеми парами “источник–назначение” с помощью матрицы трафика  $T \in \mathbf{R}^{n \times n}$ . Для любой пары различных узлов  $i, j$  обозначим через  $T_{ij} \geq 0$  трафик из узла-источника  $j$  в узел-назначение  $i$ . Трафик из узла в самого себя отсутствует; для математического удобства определим диагональные элементы матрицы трафика как  $T_{ii} = -\sum_{j \neq i} T_{ij}$  (отрицательное значение суммарного трафика с узлом-назначением  $i$ ). При таком определении диагональных элементов имеем  $T\mathbf{1} = 0$ , где  $\mathbf{1}$  – вектор, состоящий из единиц.

**Полезность сети.** Пусть  $u_{ij}$  – строго вогнутая возрастающая функция полезности для трафика из узла  $j$  в узел  $i$ , где  $j \neq i$ . Будем считать, что функции полезности дифференцируемы, с областью значений  $\mathbf{R}_{++}$  – множеством положительных чисел. (Описанные ниже методы легко распространяются на недифференцируемые функции полезности, если вместо градиентов использовать субградиенты.) Суммарная полезность, которую необходимо максимизировать, равна  $\sum_{i \neq j} u_{ij}(T_{ij})$ . Для простоты примем  $u_{ii} = 0$ , и тогда суммарную полезность можно записать как

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

Функция  $U$  имеет область значений  $\mathcal{T} = \{T \mid T_{ij} > 0 \text{ для } i \neq j\}$ , т.е. матрица трафика должна содержать положительные внедиагональные элементы.

Примерами функций полезности являются взвешенная логарифмическая полезность  $u(s) = w \log s$  и взвешенная степенная полезность  $u(s) = ws^\gamma$ , причем  $\gamma \in (0, 1)$ , а  $w > 0$  – вес.

**Матрица потоков, определяемая через узлы-назначения.** Следуя [1], агрегируем все потоки с одним и тем же узлом-назначением, рассматривая их как один товар, который сохраняется во всех узлах, кроме источника и назначения, но может быть разделен и объединен. Товарные потоки задаются матрицей потоков  $F \in \mathbf{R}^{n \times m}$  через узлы-назначения, где  $F_{i\ell} \geq 0$  – поток на ребре  $\ell$ , предназначенный для узла  $i$ . Ограничение пропускной способности ребра можно выразить как  $F^T \mathbf{1} \leq c$ , где неравенство является поэлементным. Похожая формулировка агрегирования потоков, хотя и основанная на источнике, рассмотрена в [2].

**Сохранение потока.** Поток, предназначенный для узла  $i$ , сохраняется во всех узлах  $j \neq i$ , включая дополнительное вливание трафика  $T_{ij}$ , который исходит из узла  $j$  и предназначен для узла  $i$ . Это значит, что

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

В узле-назначении весь трафик выходит, и по определению  $T_{ii}$  имеем:

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

Объединяя эти два условия и используя определение  $T_{ii}$ , сохранение потока можно компактно записать в матричной форме как

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

**Задача о многотоварных потоках.** В задаче MCF будем искать матрицу потоков, максимизирующую суммарную полезность сети при соблюдении ограничений на пропускную способность ребер и сохранение потока. Это можно выразить в виде следующей задачи:

$$(1) \quad \begin{array}{l} \text{максимизировать } U(T) \\ \text{при ограничениях } F \geq 0, \quad F^T \mathbf{1} \leq c, \quad T + FA^T = 0, \end{array}$$

с переменными  $F$  и  $T$  и неявным ограничением  $T \in \mathcal{T}$ . Здесь заданы топология сети  $A$ , пропускные способности ребер  $c$  и функции полезности трафика  $u_{ij}$ .

Можно исключить матрицу трафика  $T$  с помощью  $T = -FA^T$  и сформулировать задачу MCF только в терминах переменной  $F$ :

$$(2) \quad \begin{array}{l} \text{максимизировать } U(-FA^T) \\ \text{при ограничениях } F \geq 0, \quad F^T \mathbf{1} \leq c, \end{array}$$

с переменной  $F$  и неявным ограничением  $-FA^T \in \mathcal{T}$ . Число скалярных переменных в этой задаче равно  $nm$ . Для дальнейшего использования определим множество допустимых потоков как

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

**Существование и единственность решения.** Сначала покажем, что задача MCF (1) всегда разрешима. Рассмотрим единичный поток из каждого источника в каждое назначение по кратчайшему пути (с наименьшим числом ребер); такой путь существует, поскольку граф полносвязен. Обозначим эту матрицу потоков через  $F^{\text{sp}}$ . Теперь примем  $F = \alpha F^{\text{sp}}$ , где  $\alpha = 1/\max_{\ell}((F^{\text{sp}T} \mathbf{1})_{\ell}/c_{\ell}) > 0$ ; в результате,  $F^T \mathbf{1} \leq c$ . Очевидно, что матрица потоков  $F$  допустима; имеем  $T_{ij} = \alpha > 0$  для  $i \neq j$  и  $T = -FA^T \in \mathcal{T}$ . Это показывает, что задача всегда разрешима. Пусть  $U^{\text{sp}}$  – соответствующая целевая функция.

Можно добавить к задаче ограничение  $U(T) \geq U^{\text{sp}}$ , не меняя при этом множества решений. При таком добавлении допустимое множество становится компактным. Отсюда следует, что задача MCF (1) всегда разрешима. Решение не обязательно должно быть единственным. Однако оптимальное решение  $T$  является единственным. Отметим также, что неявное ограничение  $T = -FA^T \in \mathcal{T}$  является избыточным (см. выше).

**Решение задачи MCF.** Задача о многотоварных потоках (2) является выпуклой [3] и поэтому может быть эффективно решена. В [1] использованы

стандартные общие решатели метода внутренней точки, такие как коммерческий решатель MOSEK [4], совместно с CVXPY [5]; экземпляры задачи с десятками узлов и тысячами переменных решены за несколько секунд на одном CPU. В настоящей статье представлен алгоритм решения задачи MCF, который полностью использует возможности GPU. Для задач малой и средней размерности данный метод дает существенное ускорение по сравнению с общими методами; кроме того, он масштабируется на гораздо большие задачи, которые не могут быть решены общими методами.

## 1.2. Условие оптимальности и невязка

**Условие оптимальности.** Обозначим через  $\tilde{\mathcal{F}}$  замыкание допустимого множества, включая неявное ограничение  $T = -FA^T \in \mathcal{T}$ :

$$\tilde{\mathcal{F}} = \mathcal{F} \cap \{F \mid -FA^T \in \text{cl}(\mathcal{T})\},$$

где  $\text{cl}(\mathcal{T})$  – замыкание  $\mathcal{T}$ .

Матрица потоков  $F$  оптимальна для (2) тогда и только тогда, когда  $F \in \mathcal{F}$ ,  $-FA^T \in \mathcal{T}$ , и неравенство

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

справедливо для всех  $Z \in \tilde{\mathcal{F}}$ , где  $G = \nabla_F(-U)(-FA^T)$  (например, см. [3, раздел 4.2.3]). Имеем  $G = U'A$ , где  $U'_{ij} = u'_{ij}((-FA^T)_{ij})$ .

**Условие оптимальности через проекцию на  $\mathcal{F}$ .** Для дальнейшего использования выразим вышеприведенное условие оптимальности в терминах проекции матрицы  $Q$  на  $\mathcal{F}$ . Обозначим через  $\Pi$  евклидову проекцию на  $\mathcal{F}$ . Предположим, что  $Q \in \mathbf{R}^{n \times m}$ , и зададим  $F = \Pi(Q)$ ; в результате,  $F \in \mathcal{F}$ . Предположим также, что  $-FA^T \in \mathcal{T}$ , так что  $G = \nabla_F((-U)(-FA^T))$  существует. Тогда  $F$  также является евклидовой проекцией  $Q$  на  $\tilde{\mathcal{F}}$ . Отсюда следует, что  $\text{Tr}(Z - F)^T G \geq 0$  для всех  $Z \in \tilde{\mathcal{F}}$ , поэтому условие оптимальности выше выполняется, и матрица потоков  $F$  оптимальна. Очевидно, оно будет выполняться и при более слабом условии:

$$G = \gamma(F - Q) \text{ для некоторого } \gamma \geq 0.$$

Подведем итог:  $F$  оптимальна, если  $F = \Pi(Q)$  для некоторого  $Q$ ,  $-FA^T \in \mathcal{T}$  и  $G = \gamma(F - Q)$  для некоторого  $\gamma \geq 0$ . Обратное утверждение также верно: если  $F$  оптимальна, то  $F = \Pi(Q)$  для некоторого  $Q$  с  $-FA^T \in \mathcal{T}$  и  $G = \gamma(F - Q)$  для некоторого  $\gamma \geq 0$ . (Действительно, это справедливо при  $\gamma = 1$  и  $Q = F - G$ .) Это условие оптимальности легко интерпретируется:  $F$  является неподвижной точкой операции проекции градиента с длиной шага  $\gamma$ .

**Невязка оптимальности.** Для любого  $Q \in \mathbf{R}^{n \times m}$  с  $F = \Pi(Q)$  определим невязку (оптимальности) как

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

где  $\|\cdot\|_F^2$  обозначает квадрат фробениусовой нормы матрицы (сумму квадратов ее элементов). При  $-FA^T \in \mathcal{T}$  правая часть является квадратичной функцией  $\gamma$ , поэтому минимум легко выражается в явном виде как

$$(3) \quad r(Q) = \begin{cases} \|G\|_F^2 - \frac{\text{Tr}^2 G^T (F - Q)}{\|F - Q\|_F^2}, & \text{если } -FA^T \in \mathcal{T}, F \neq Q, \\ & \text{Tr} G^T (F - Q) \geq 0, \\ \|G\|_F^2, & \text{если } -FA^T \in \mathcal{T}, F = Q \text{ или} \\ & \text{Tr} G^T (F - Q) < 0, \\ \infty & \text{в противном случае.} \end{cases}$$

Очевидно, что  $F = \Pi(Q)$  оптимальна тогда и только тогда, когда  $r(Q) = 0$ .

### 1.3. Обзор имеющихся результатов

**Многоотоварный сетевой поток.** Различные постановки задач MCF были сформулированы и изучены ранее. Начало положено в публикациях [6, 7], посвященных версии с линейными функциями полезности, которая может быть сформулирована как задача линейного программирования (ЛП). В более поздних работах предложены нелинейные выпуклые постановки [8, 9] и (невыпуклые) смешанные целочисленные формулировки [10–12] задач MCF для различных приложений. Они широко используются в управлении перевозками [13–15], энергетике и экономике [8, 10, 16] и сетевых коммуникациях [11, 17, 18]. В обзорной статье [19] описано более двухсот исследований по задачам MCF за 2000–2019 гг. В настоящей статье рассматривается нелинейная выпуклая постановка задач MCF и разрабатываются GPU-совместимые алгоритмы для решения экземпляров таких задач большой размерности. Обзор нелинейных выпуклых задач MCF можно найти в [20]. Совсем недавно модели MCF стали использоваться при составлении расписаний коммуникации между несколькими GPU для задач глубокого обучения [21, 22], но в соответствующих задачах MCF применяются решатели на основе CPU.

**Методы первого порядка для выпуклой оптимизации.** Методы первого порядка, такие как алгоритм градиентного спуска, алгоритм проксимальной точки, прямо-двойственный гибридный градиентный алгоритм и их ускоренные версии, применяются для решения различных задач выпуклой оптимизации. По сравнению с методами второго порядка, использующими информацию о гессиане, методы первого порядка известны своей низкой вычислительной сложностью и поэтому привлекательны для решения задач оптимизации большой размерности. Недавно прямо-двойственный гибридный градиентный алгоритм был исследован для решения больших задач ЛП [23–25] и задач оптимальной транспортировки [26] на GPU. Другие методы первого порядка, такие как ADMM (Alternating Direction Method of Multipliers), использованы для разработки методов оптимизации с GPU-ускорением для задач об оптимальном потоке мощности (optimal power flow) [27, 28].

**Оптимизация сетевого потока с GPU-ускорением.** В статье [29], посвященной методам на базе GPU для оптимизации сетевых потоков, рассмотрена реализация параллельного алгоритма маршрутизации на GPU для программно-определяемых сетей (software-defined networking, SDN), который решает лагранжеву релаксацию задачи смешанного целочисленного ЛП. Авторы [30] реализовали генетический метод на GPU для решения задачи маршрутизации, сформулированной в виде целочисленной задачи ЛП. В [31] рассмотрена ЛП-формулировка задачи о многотоварных сетевых потоках и построена модель глубокого обучения для генерации новых столбцов в методе отложенной генерации столбцов. Авторы [32] реализовали асинхронный алгоритм push-relabel для задачи о максимальном сетевом потоке с одним товаром, который является гибридом CPU-GPU. В соответствии с [1] в статье [33] использована точно такая же формулировка агрегирования потоков MCF, что и в настоящей работе; обучена нейросетевая модель для минимизации лагранжевой релаксации без ограничений задачи, а результат передан в качестве теплого старта в решатель Gurobi [34] для получения окончательного ответа. Авторы [35] встроили формулировку задачи агрегирования многотоварных потоков на основе узлов-источников в решение комбинированной транспортной модели и использовали ускоренный вариант проксимального альтернирующего алгоритма “предиктор-корректор” (proximal alternating predictor-corrector algorithm). По их утверждению, предложенный алгоритм ориентирован на GPU, но численные эксперименты приведены на CPU на примерах сетей небольшого размера. В [36] использован прямо-двойственный градиентный метод для решения комбинированных моделей трафика, который, однако, не ориентирован на GPU.

#### *1.4. Новизна*

В настоящей работе, мотивированной последними достижениями в области методов оптимизации на основе GPU, решение нелинейных выпуклых задач MCF большой размерности ускоряется с помощью GPU. В частности, используется формулировка задачи MCF из [1] (также описанная выше), которая компактно представлена в матричном виде и требует меньшего количества переменных за счет агрегирования потоков. Показывается, что данная конкретная постановка задачи может быть эффективно решена с помощью прямо-двойственного гибридного градиентного метода первого порядка при работе на GPU.

Насколько известно авторам, настоящая статья является первой, посвященной точному решению выпуклых задач MCF на GPU. В классических работах по таким задачам MCF большой размерности обычно используется лагранжева релаксация для связанных ограничений, а полученные подзадачи меньшей размерности решаются параллельно (например, см. [20]). В настоящей работе не используется какая-либо явная стратегия декомпозиции задачи, а алгоритмическое ускорение является в основном эмпирическим и зависит от высокооптимизированных ядер CUDA для матричных операций.

Более того, размерность задачи уменьшается за счет агрегирования потоков. Поэтому представленный здесь метод имеет более простую форму, не требующую массового решения подзадач и синхронизации, к тому же он является точным.

### 1.5. План дальнейшего изложения

Предлагаемый алгоритм описывается в разделе 2. Экспериментальные результаты, использующие предложенную реализацию PyTorch, приводятся и обсуждаются в разделе 3; очень похожие результаты, полученные с помощью предложенной реализации JAX, даны в Приложении 7. Итоги настоящей работы подводятся в разделе 4. Код и все данные, необходимые для воспроизведения представленных здесь результатов, можно найти по адресу

<https://github.com/cvxgrp/pdmcf>

## 2. Прямо-двойственный гибридный градиентный метод

### 2.1. Прямо-двойственная формулировка с седловой точкой

Выведем прямо-двойственную формулировку с седловой точкой для задачи MCF (1). Обозначим через  $\mathcal{I}$  индикаторную функцию множества  $\mathcal{F}$ , т.е.  $\mathcal{I}(F) = 0$  для  $F \in \mathcal{F}$  и  $\mathcal{I}(F) = \infty$  в противном случае. Перейдем к минимизации  $-U$  в (1), чтобы получить эквивалентную задачу

$$(4) \quad \begin{aligned} & \text{минимизировать} \quad -U(T) + \mathcal{I}(F) \\ & \text{при ограничении} \quad T = -FA^T \end{aligned}$$

с переменными  $T$  и  $F$ . Введем двойственную переменную  $Y \in \mathbf{R}^{n \times n}$ , связанную с (матричным) ограничением-равенством. Тогда лагранжиан имеет вид

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

(см. [3, глава 5]). Лагранжиан  $\mathcal{L}$  является выпуклым по прямым переменным  $(T, F)$  и аффинным (следовательно, вогнутым) по двойственной переменной  $Y$ . Если  $(T, F; Y)$  – седловая точка  $\mathcal{L}$ , то  $(T, F)$  является решением задачи (4) (а  $F$  – решением задачи MCF (2)); обратное утверждение также справедливо.

Можно аналитически минимизировать  $\mathcal{L}$  на  $T$ , чтобы получить редуцированный лагранжиан

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

где  $U^*$  – сопряженная функция к  $U$  [3, раздел 3.3]. Этот редуцированный лагранжиан является выпуклым по прямой переменной  $F$  и вогнутым по двойственной переменной  $Y$ . Если  $(F; Y)$  – седловая точка  $\hat{\mathcal{L}}$ , то  $F$  – решение задачи MCF (2) (см. [37, раздел 1]). Заметим, что  $\hat{\mathcal{L}}$  является выпукло-вогнутым, с билинейной связью.

## 2.2. Основной метод PDHG

Прямо-двойственный гибридный градиентный алгоритм (PDHG), впервые представленный в [38] и позже изученный в [39, 40], является методом первого порядка для нахождения седловой точки выпукло-вогнутой функции с билинейной связью. В [40, раздел 4.1] алгоритм расширен за счет включения избыточной релаксации (перерелаксации), которая, как было замечено, улучшает сходимость на практике. Для (5) алгоритм PDHG имеет вид

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

где  $\mathbf{prox}_f(v) = \operatorname{argmin}_x (f(x) + (1/2)\|x - v\|_2^2)$  обозначает проксимальный оператор для  $f$  [41],  $\alpha, \beta > 0$  – положительные длины шага, удовлетворяющие  $\alpha\beta \leq 1/\|A\|_2^2$ , а  $\rho \in (0, 2)$  представляет собой параметр перерелаксации.

Разумными вариантами выбора параметров являются

$$\alpha = \beta = 1/\|A\|_2, \quad \rho = 1,9.$$

(Вместо  $\|A\|_2$  можно использовать верхнюю границу для  $\|A\|_2$ .)

**Сходимость.** В статье [40] показано, что при существовании седловой точки функции  $\hat{\mathcal{L}}$  последовательность  $(F^k; Y^k)$  сходится к седловой точке функции  $\hat{\mathcal{L}}$  при  $k \rightarrow \infty$ . Как известно, для MCF существуют оптимальная матрица потока и двойственная переменная, поэтому  $F^k$  сходится к оптимальной матрице потока. Отсюда следует, что  $r(F^{k-1/2} + \alpha Y^k A) \rightarrow 0$  при  $k \rightarrow \infty$ . Заметим, что  $-FA^T \in \mathcal{T}$  выполняется только в пределе.

## 2.3. Проксимальные операторы

Здесь более подробно рассмотрим два проксимальных оператора, присутствующих в PDHG.

**Первый проксимальный оператор.** Отметим, что  $\mathbf{prox}_{\alpha\mathcal{T}}$ , применяемый на шаге обновления  $\hat{F}^{k+1/2}$  в (6), является проекцией на  $\mathcal{F}$ :

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

Поскольку ограничения, определяющие  $\mathcal{F}$ , разделяются по столбцам матрицы  $F$ , можно вычислить  $\Pi(F)$ , спроецировав каждый столбец  $f_\ell$  из  $F$  на масштабированный симплекс  $\mathcal{S}_\ell = \{f \mid f \geq 0, \mathbf{1}^T f \leq c_\ell\}$ . Эта проекция имеет вид

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

где  $\mu_\ell$  – оптимальный множитель Лагранжа, а оператор  $(a)_+ = \max\{a, 0\}$  применяется к вектору поэлементно. Оптимальное значение  $\mu_\ell$  – наименьшее

неотрицательное значение, для которого  $(f_\ell - \mu_\ell \mathbf{1})_+^T \mathbf{1} \leq c_\ell$ . Его легко найти с помощью алгоритма биссекции; см. раздел 2.6.

**Второй проксимальный оператор.** Проксимальный оператор, применяемый на шаге обновления  $\hat{Y}^{k+1}$  в (6), может быть разложен покомпонентно, поскольку  $\beta(-U)^*$  является суммой функций различных переменных. (Диагональные элементы  $-u_{ii}$  равны нулю, поэтому  $(-\beta u_{ii})^*$  является индикаторной функцией множества  $\{0\}$ , а ее проксимальный оператор – нулевой функцией.) Для каждого внедиагонального элемента  $i \neq j$  необходимо оценить

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

Эти одномерные проксимальные операторы легко вычисляются в общем случае. Для взвешенной логарифмической полезности  $u(s) = w \log s$  имеем

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

Для взвешенной степенной полезности  $u(s) = ws^\gamma$  оператор  $\mathbf{prox}_{\beta(-u)^*}(y)$  – единственное отрицательное число  $z$ , для которого

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

где

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

#### 2.4. Адаптивный выбор длины шага

В базовом алгоритме PDHG (6) шаги  $\alpha$  и  $\beta$  фиксированы. Было замечено [23], что на практике адаптивное изменение их длины в процессе работы алгоритма может существенно улучшить сходимость. Опишем реализацию адаптивного выбора длины шага.

Выразим длину шага как

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

где  $\eta \leq 1/\|A\|_2$  и  $\omega^k > 0$  задает вес прямых переменных. При  $\omega^k = 1$  получаем базовый PDHG (6).

Вес  $\omega^k$  инициализируется значением  $\omega^0 = 1$  и затем адаптируется следующим образом [23, раздел 3.3]:

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

где  $\Delta_F^{k+1} = \|F^{k+1/2} - F^{k-1/2}\|_F$ ,  $\Delta_Y^{k+1} = \|Y^{k+1} - Y^k\|_F$  и  $\theta$  – параметр, равный 0,5 в данной реализации. Интуиция, лежащая в основе обновления веса (7), состоит в балансе невязок прямых и двойственных переменных; подробности см. в [23, раздел 3.3]. В [23]  $\omega$  обновляется при каждом перезапуске. В настоящей статье без использования перезапусков обнаружено, что

обновление  $\omega^k$  каждые  $k^{\text{adapt}}$  итераций при одновременном выполнении условий  $\Delta_F^k > 10^{-5}$  и  $\Delta_Y^k > 10^{-5}$  хорошо работает на практике для задачи MCF. В экспериментах использовано  $k^{\text{adapt}} = 100$ . Также можно прекратить адаптацию  $\omega^k$  после некоторого количества итераций, сохраняя его значение постоянным в последующих итерациях. По крайней мере, технически это означает, что доказательство сходимости при постоянном  $\omega$  справедливо и для адаптивного алгоритма.

**Простая граница для  $\|A\|_2$ .** Можно легко вычислить простую верхнюю границу для

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

где  $\lambda_{\max}$  обозначает максимальное собственное значение. Заметим, что  $AA^T$  – матрица Лапласа, связанная с сетью; как известно, для нее имеет место верхняя граница

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

где  $d_{\max}$  – наибольшая степень узла в графе. (Для полноты этот результат строго доказан в Приложении 6.) Таким образом, можно взять

$$(8) \quad \eta = 1/\sqrt{2d_{\max}}.$$

## 2.5. Алгоритм

Представим в окончательной форме алгоритм, который будем называть PDMCF. Задаем  $r^0 = +\infty$ ,  $\alpha^0 = \eta/\omega^0$  и  $\beta^0 = \eta\omega^0$ , где  $\eta$  определена в (8), а  $\omega^0 = 1$ .

### Алгоритм 2.1. PDMCF

**При заданных  $F^{-1/2}$ ,  $Y^0$  и параметре  $\epsilon > 0$**

**для  $k = 0, 1, \dots$ :**

1. *Проверить критерий остановки.* Выйти и вернуть  $\hat{F}^{k-1/2}$  при выполнении условия  $r^k < n\epsilon$ ;
2. *Базовые обновления PDHG (6):*

$$\begin{aligned} \hat{F}^{k+1/2} &= \Pi(F^{k-1/2} + \alpha^k Y^k A), \\ F^{k+1} &= 2\hat{F}^{k+1/2} - F^{k-1/2}, \\ \hat{Y}_{ij}^{k+1} &= \begin{cases} \text{prox}_{\beta^k(-u_{ij})^*}(Y_{ij}^k - \beta^k(F^{k+1}A^T)_{ij}), & j \neq i, \\ 0, & j = i, \end{cases} \\ F^{k+1/2} &= \rho\hat{F}^{k+1/2} + (1-\rho)F^{k-1/2}, \\ Y^{k+1} &= \rho\hat{Y}^{k+1} + (1-\rho)Y^k; \end{aligned}$$
3. *Адаптивные обновления длины шага (7)*  
(если  $k$  кратно  $k^{\text{adapt}}$  и  $\Delta_F^{k+1}, \Delta_Y^{k+1} > \tau$ ):
$$\begin{aligned} \omega^{k+1} &= \left(\Delta_Y^{k+1}/\Delta_F^{k+1}\right)^\theta (\omega^k)^{1-\theta}, \\ \alpha^{k+1} &= \eta/\omega^{k+1}, \quad \beta^{k+1} = \eta\omega^{k+1}. \end{aligned}$$

**Инициализация.** Всегда выбираем  $F^{-1/2} = 0$  и  $Y^0 = I - \mathbf{1}\mathbf{1}^T$ . В качестве альтернативы можно использовать более подходящие значения  $F^{-1/2}$  и  $Y$ , например, при теплом старте, когда задача с похожими данными уже решена. Подробнее об этом расскажем в разделе 3.1.

**Критерий останова.** Поскольку  $\hat{F}^{k+1/2}$  является результатом проекции на  $\mathcal{F}$ , невязка оптимальности (3) имеет вид

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

Используем критерий останова  $r^k < nm\epsilon$ , т.е. поэлементная нормализованная невязка  $r^k/nm$  должна быть меньше заданного пользователем порога  $\epsilon$ .

## 2.6. Детали реализации

**Индексация матрицы инцидентности.** Храним только индексы ненулевых элементов матрицы  $A$ . Умножение матриц на  $A$  и  $A^T$  можно эффективно вычислить с помощью функций `scatter` и `gather`, которые являются высокооптимизированными ядрами CUDA и доступны в большинстве основных вычислительных языков GPU.

**Проекция на масштабированный симплекс.** Для вычисления  $\mu_\ell$  при  $(f_\ell)_+^T \mathbf{1} > c_\ell$ , следуя [42], сначала сортируем  $f_\ell$  от наибольшего элемента к наименьшему, чтобы сформировать  $f'_\ell$ . Затем находим наибольший индекс  $t$ , при котором  $f'_{\ell t} - ((\sum_{i=1}^t f'_{\ell i} - c_\ell)/t) > 0$ . Наконец, берем  $\mu_\ell = (\sum_{i=1}^t f'_{\ell i} - c_\ell)/t$ .

В недавних работах предложен более эффективный метод вычисления проекции на симплексное множество (например, см. [43, 44]). Для простоты реализации в настоящей статье используется более простой алгоритм, описанный выше.

## 3. Эксперименты

Все эксперименты были выполнены на одном графическом процессоре H100 с 80 Гб памяти, поддерживаемом 26 виртуальными ядрами CPU и 241 Гб оперативной памяти. Ниже представлены результаты для предложенной реализации PyTorch; аналогичные результаты, описанные в Приложении 7, получены с помощью предложенной реализацией JAX.

### 3.1. Примеры

**Данные и параметры.** Рассматривались взвешенные логарифмические полезности вида  $u_{ij}(T_{ij}) = w_{ij} \log T_{ij}$ . Величина  $\log w_{ij}$  принималась равномерной на  $[\log 0,3, \log 3]$ . Для топологии сети сначала создавались  $n$  двумерных точек данных  $\xi_i \in \mathbf{R}^2$ , каждая из которых обозначалась как  $(\xi_{ix}, \xi_{iy})$  для  $i = 1, \dots, n$ . Величины  $\xi_{ix}$  и  $\xi_{iy}$  принимались равномерными на  $[0, 1]$ . Затем добавлялись оба ребра  $(\xi_i, \xi_j)$  и  $(\xi_j, \xi_i)$ , когда либо  $\xi_i$  входит в число  $q$ -ближайших соседей  $\xi_j$ , либо наоборот. Для каждого ребра  $\ell$  задавалась его пропускная способность  $c_\ell$ , где величина  $\log c_\ell$  принималась равномерной на  $[\log 0,5, \log 5]$ .

Использовался порог критерия остановки  $\epsilon = 0,01/(n(n - 1))$  для задач малой и средней размерности и  $\epsilon = 0,03/(n(n - 1))$  для задач большой размерности. Результаты сравнивались с коммерческим решателем MOSEK, работающим на CPU, с настройками по умолчанию. MOSEK способен решать задачи с высокой точностью; анализ показал, что для всех экземпляров задач нормированная разность полезностей между результатами PDMCF и MOSEK не превышает примерно 0,01. Парные нормализованные (оптимальные) полезности лежали в диапазоне от 1 до 10. То есть PDMCF находит потоки, которые являются от 0,1% до 1% субоптимальными по сравнению с потоками, найденными MOSEK.

**Задачи малой и средней размерности.** В табл. 1 показано время работы MOSEK и PDMCF, необходимое для решения задач различных размерностей. В столбце  $nm$  указано количество скалярных переменных оптимизации в экземпляре задачи. Видно, что предложенная реализация PDMCF на GPU дает ускорение по сравнению с MOSEK от 10 до 1000 раз, причем более значительное ускорение наблюдается для экземпляров задач большей размерности. Также приведено время работы PDMCF на CPU, которое все еще ниже, чем у MOSEK, но со значительно меньшим ускорением. Аналогичная производительность наблюдается и для предложенной реализации JAX (см. Приложение 7).

**Таблица 1.** Время работы для задач малой и средней размерности

| Размерность задачи |     |        |                 | время (с) |             |             |          |
|--------------------|-----|--------|-----------------|-----------|-------------|-------------|----------|
| $n$                | $q$ | $t$    | $nm$            | MOSEK     | PDMCF (CPU) | PDMCF (GPU) | итерации |
| 100                | 10  | 1178   | $1 \times 10^5$ | 5         | 1           | 0,5         | 490      |
| 200                | 10  | 2316   | $5 \times 10^5$ | 23        | 2           | 0,7         | 690      |
| 300                | 10  | 3472   | $1 \times 10^6$ | 95        | 6           | 0,8         | 840      |
| 500                | 10  | 5738   | $3 \times 10^6$ | 340       | 18          | 1,1         | 950      |
| 500                | 20  | 11 176 | $6 \times 10^6$ | 1977      | 34          | 1,4         | 790      |
| 1000               | 10  | 11 424 | $1 \times 10^7$ | 2889      | 1382        | 19,5        | 7220     |
| 1000               | 20  | 22 286 | $2 \times 10^7$ | 16 765    | 349         | 5,1         | 1040     |

**Задачи большой размерности.** В табл. 2 показано время работы для нескольких экземпляров задач большой размерности. MOSEK не может решить все эти задачи из-за ограничений памяти. PDMCF справляется со всеми этими задачами, причем самая большая из них содержит  $10^9$  переменных.

**Таблица 2.** Время работы для задач большой размерности

| Размерность задачи |     |         |                 | время (с) |             |             |          |
|--------------------|-----|---------|-----------------|-----------|-------------|-------------|----------|
| $n$                | $q$ | $t$     | $nm$            | MOSEK     | PDMCF (CPU) | PDMCF (GPU) | итерации |
| 3000               | 10  | 34 424  | $1 \times 10^8$ | OOM       | 7056        | 96          | 4140     |
| 5000               | 10  | 57 338  | $3 \times 10^8$ | OOM       | 19 152      | 395         | 3970     |
| 10000              | 10  | 114 054 | $1 \times 10^9$ | OOM       | 87 490      | 1908        | 4380     |

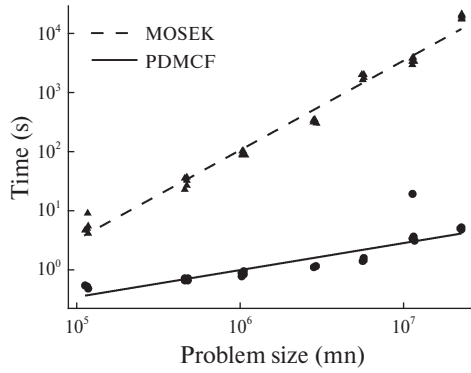


Рис. 1. Время работы для задач малой и средней размерности.

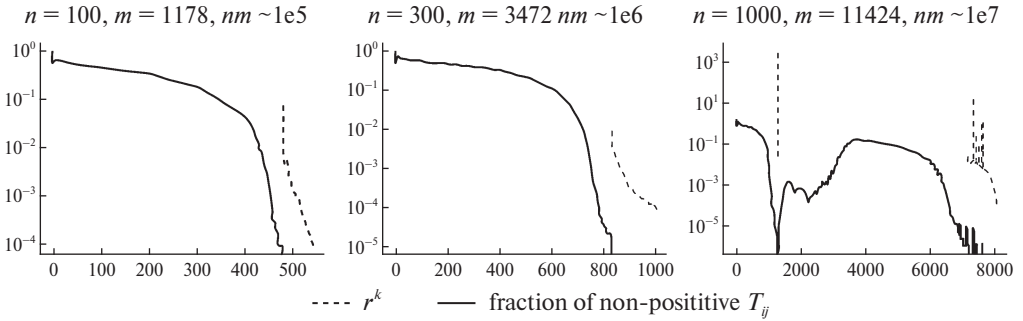


Рис. 2. График сходимости для задач малой и средней размерности.

**Масштабирование.** На рис. 1 представлена диаграмма времени работы для экземпляров задач малой и средней размерности. Здесь взяты 5 экземпляров задач, сгенерированных на случайной выборке из  $\{0, 1, 2, 3, 4\}$  для различных значений  $n, q$ , перечисленных в табл. 1. Ось  $x$  соответствует размерности переменной оптимизации  $nm$ , а ось  $y$  – времени работы в секундах (в логарифмической шкале). Линиями показаны аффинные функции для этих данных, с наклоном около 1,5 для MOSEK и около 0,5 для PDMCF.

**График сходимости.** На рис. 2 показана сходимость для трех экземпляров задачи с размерностями  $10^5, 10^6$  и  $10^7$  с PDMCF, где ось  $x$  соответствует числу итераций. Особенно на начальных итерациях наблюдается бесконечная невязка  $r^k$ , так как  $-FA^T \notin \mathcal{T}$ . Для этих итераций сплошной линией показана доля неположительных внедиагональных элементов в  $T$ . Для допустимых итераций прерывистой линией показана (конечная) невязка.

**Теплый старт.** В разделе 2.5 начинаем с некоторых простых начальных  $F^{-1/2}$  и  $Y^0$ . Также тестируем работу PDMCF с теплым стартом. На рис. 3 показано, как меняется время работы при различных теплых стартах. Чтобы сформировать эти старты, для некоторого коэффициента возмущения  $\nu$  случайным образом возмущаем элементы весовой матрицы полезности, чтобы получить  $\tilde{w}_{ij} = (1 \pm \nu)w_{ij}$ , каждую с вероятностью 1/2. Решаем задачу о мно-

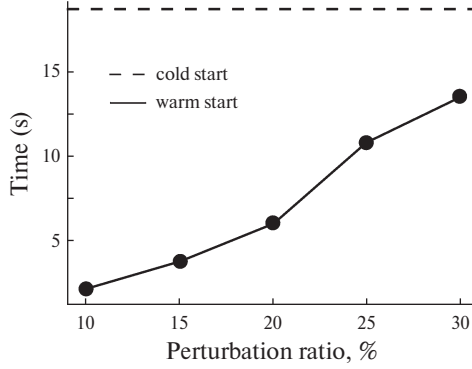


Рис. 3. График теплового старта для задачи средней размерности.

готоварных сетевых потоках с возмущенным весом полезности  $\tilde{w}$  с помощью PDMCF до тех пор, пока не попадем в допустимую точку  $(F^{\text{feas}}, Y^{\text{feas}})$ , удовлетворяющую  $(-F^{\text{feas}} A^T)_{ij} > 0$  для всех различных  $i, j$ . Записываем прямой вес в этой точке как  $\omega^{\text{feas}}$ . Затем решаем желаемую задачу о многотоварных сетевых потоках с исходным весом полезности  $w$  с  $F^{-1/2} = F^{\text{feas}}, Y^0 = Y^{\text{feas}}$  и  $\omega^0 = \omega^{\text{feas}}$ .

Отметим, что настройка  $\omega^0 = \omega^{\text{feas}}$  важна для ускорения сходимости, иначе обычно требуется такое же количество итераций для сходимости, как и при холодном старте, если просто выбрать  $\omega^0 = 1$ . На рис. 3 взят экземпляр задачи с  $n = 1000, q = 10$ . Ось  $x$  соответствует коэффициенту возмущения  $\nu$ , а ось  $y$  – времени работы в секундах. Как видно, при коэффициенте возмущения  $\nu = 10\%$  экономия времени работы составляет более 80%. При  $\nu = 30\%$  эта экономия снижается примерно до 30%, что вполне логично, учитывая, что при большем возмущении веса полезности исходной задачи и возмущенной задачи отличаются. Поэтому ожидается, что теплый старт будет находиться дальше от оптимального решения исходного экземпляра задачи.

#### 4. Заключение

В работе представлен алгоритм PDMCF, который ускоряет решение задач о многотоварных сетевых потоках на графических процессорах (GPU). Данный метод исходит из формулировки задачи о многотоварных сетевых потоках на основе узлов-назначений, что позволяет уменьшить количество переменных оптимизации по сравнению с классической формулировкой задачи. Затем для решения так сформулированной задачи применяется алгоритм PDHG. Эмпирические результаты подтверждают, что предложенный алгоритм дружелюбен к GPU и ускоряет вычисления (сокращает время работы) на три порядка по сравнению с классическими коммерческими решателями на базе CPU. Кроме того, этот алгоритм способен решать в десять раз большие задачи, чем те, которые могут быть решены коммерческими решателями на базе CPU.

## Благодарности

Авторы благодарят Энтони Деглериса и Партха Нобеля за ценные обсуждения деталей реализации, а также Демьяна Ярмошика за очень полезные отзывы, которые помогли переработать первоначальную рукопись.

### ПРИЛОЖЕНИЕ 1. Верхняя граница для $\lambda_{\max}(AA^T)$

Для направленного графа с матрицей инцидентности  $A$  обозначим через  $d_i = (AA^T)_{ii}$  степень вершины  $i$ , а для  $i \neq j$  величина  $-(AA^T)_{ij}$  есть количество ребер, соединяющих вершины  $i$  и  $j$ , т.е. два, если оба ребра  $(i, j)$  и  $(j, i)$  существуют. Заметим, что  $\lambda_{\max}(AA^T) = \max_{\|x\|_2=1} x^T(AA^T)x = \max_{x \neq 0} \frac{x^T(AA^T)x}{x^T x}$ . Имеем

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

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

### ПРИЛОЖЕНИЕ 2. Результаты JAX

Результаты, представленные в разделе 3.1, относятся к предложенной реализации PyTorch. Здесь приводятся те же результаты для предложенной реализации JAX. В табл. 3 и 4 показано время работы на тех же экземплярах задачи, что и в табл. 1 и 2. Отметим, что JIT-компиляция (just-in-time) JAX добавляет накладные расходы на компиляцию функций в первый раз, поэтому на задачах малой размерности она работает хуже, чем ее аналог PyTorch. Время работы этих двух версий близко для задач средней и большой размерности, причем JAX работает несколько медленнее.

**Таблица 3.** Время работы для задач малой и средней размерности (JAX)

| Размерность задачи |     |        |                 | время (с) |             |             | итерации |
|--------------------|-----|--------|-----------------|-----------|-------------|-------------|----------|
| $n$                | $q$ | $m$    | $nm$            | MOSEK     | PDMCF (CPU) | PDMCF (GPU) |          |
| 100                | 10  | 1178   | $1 \times 10^5$ | 5         | 12          | 5           | 490      |
| 200                | 10  | 2316   | $5 \times 10^5$ | 23        | 57          | 6           | 690      |
| 300                | 10  | 3472   | $1 \times 10^6$ | 95        | 164         | 6           | 840      |
| 500                | 10  | 5738   | $3 \times 10^6$ | 340       | 548         | 7           | 950      |
| 500                | 20  | 11 176 | $6 \times 10^6$ | 1977      | 890         | 8           | 790      |
| 1000               | 10  | 11 424 | $1 \times 10^7$ | 2889      | 18 554      | 26          | 7150     |
| 1000               | 20  | 22 286 | $2 \times 10^7$ | 16 765    | 5143        | 15          | 1040     |

**Таблица 4.** Время работы для задач большой размерности (JAX)

| Размерность задачи |     |         |                 | время (с) |             |             | итерации |
|--------------------|-----|---------|-----------------|-----------|-------------|-------------|----------|
| $n$                | $q$ | $m$     | $nm$            | MOSEK     | PDMCF (CPU) | PDMCF (GPU) |          |
| 3000               | 10  | 34 424  | $1 \times 10^8$ | OOM       | 10 6274     | 139         | 4140     |
| 5000               | 10  | 57 338  | $3 \times 10^8$ | OOM       | 382 400     | 421         | 3970     |
| 10 000             | 10  | 114 054 | $1 \times 10^9$ | OOM       | 1 809 517   | 2078        | 4380     |

## СПИСОК ЛИТЕРАТУРЫ

1. Yin P., Diamond S., Lin B., Boyd S. Network optimization for unified packet and circuit switched networks // ArXiv Preprint ArXiv:1808.00586. 2019. <https://arxiv.org/abs/1808.00586>
2. Bar-Gera H., Boyce D. Origin-based algorithms for combined travel forecasting models // Transportation Research Part B: Methodological. 2003. V. 37. No. 5. P. 405–422.
3. Boyd S., Vandenberghe L. Convex Optimization. Cambridge: Cambridge University Press, 2004.
4. ApS MOSEK. The MOSEK optimization toolbox for MATLAB manual. Version 9.0. 2019. <http://docs.mosek.com/9.0/toolbox/index.html>
5. Diamond S., Boyd S. CVXPY: A Python-embedded modeling language for convex optimization // J. Machin. Learn. Res. 2016. V. 17. No. 83. P. 1–5.
6. Ford Jr.L., Fulkerson D. A suggested computation for maximal multi-commodity network flows // Management Science. 1958. V. 5. No. 1. P. 97–101.
7. Hu T. Multi-commodity network flows // Operations Research. 1963. V. 11. No. 3. P. 344–360.
8. Gautier A., Granot F. Forest management: A multicommodity flow formulation and sensitivity analysis // Management Science. 1995. V. 41. No. 10. P. 1654–1668.
9. Ouorou A., Mahey P. A minimum mean cycle cancelling method for nonlinear multicommodity flow problems // Eur. J. Oper. Res. 2000. V. 121. No. 3. P. 532–548.
10. Manfren M. Multi-commodity network flow models for dynamic energy management—mathematical formulation // Energy Procedia. 2012. V. 14. P. 1380–1385.
11. Kabadurmus O., Smith A. Multi-commodity k-splittable survivable network design problems with relays // Telecommunication Systems. 2016. V. 62. P. 123–133.

12. *Zantuti A.* The capacity and non-simultaneously multicommodity flow problem in wide area network and data flow management // Proceedings of the 18th International Conference on Systems Engineering. 2005. P. 76–80. <https://doi.org/10.1109/ICSENG.2005.81>
13. *Erera A., Morales J., Savelsbergh M.* Global intermodal tank container management for the chemical industry // Transportation Research Part E: Logistics and Transportation Review. 2005. V. 41. P. 551–566.
14. *Mesquita M., Moz M., Paias A., Pato M.* A decompose-and-fix heuristic based on multi-commodity flow models for driver rostering with days-off pattern // European Journal of Operational Research. 2015. V. 245. No. 2. P. 423–437.
15. *Rudi A., Frohling M., Zimmer K., Schultmann F.* Freight transportation planning considering carbon emissions and in-transit holding costs: a capacitated multi-commodity network flow model // EURO J. Transport. Logist. 2016. V. 5. P. 123–160. <https://api.semanticscholar.org/CorpusID:53229695>
16. *Singh I.* A dynamic multi-commodity model of the agricultural sector: A regional application in Brazil // European Economic Review. 1978. V. 11. P. 155–179. <https://api.semanticscholar.org/CorpusID:152973282>
17. *Wagner D., Raidl G., Pferschy U., Mutzel P., Bachhiesl P.* A multi-commodity flow approach for the design of the last mile in real-world fiber optic networks // Operation Research Proceedings. 2006. <https://api.semanticscholar.org/CorpusID:17037020>
18. *Layeb S., Heni R., Balma A.* Compact MILP models for the discrete cost multicommodity network design problem // 2017 International Conference on Engineering & MIS. IEEE, 2017. P. 1–7.
19. *Salimifard K., Bigharaz S.* The multicommodity network flow problem: state of the art classification, applications, and solution methods // Operational Research. 2022. V. 22. No. 1. P. 1–47.
20. *Ouorou A., Mahey P., Vial J.* A survey of algorithms for convex multicommodity flow problems // Management Science. 2000. V. 46. No. 1. P. 126–147.
21. *Liu X., Arzani B., Kakarla S., Zhao L., Liu V., Castro M., Kandula S., Marshall L.* Rethinking machine learning collective communication as a multi-commodity flow problem // Proceedings of the ACM SIGCOMM 2024 Conference. 2024. P. 16–37.
22. *Basu P., Zhao L., Fantl J., Pal S., Krishnamurthy A., Khoury J.* Efficient all-to-all collective communication schedules for direct-connect topologies // Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing. 2024. P. 28–41. <https://doi.org/10.1145/3625549.3658656>
23. *Applegate D., Díaz M., Hinder O., Lu H., Lubin M., O’Donoghue B., Schudy W.* Practical large-scale linear programming using primal-dual hybrid gradient // Neural Information Processing Systems. 2021. <https://api.semanticscholar.org/CorpusID:235376806>
24. *Lu H., Yang J.* cuPDLP.jl: A GPU implementation of restarted primal-dual hybrid gradient for linear programming in Julia // ArXiv Preprint ArXiv:2311.12180. 2024. <https://arxiv.org/abs/2311.12180>
25. *Lu H., Peng Z., Yang J.* MPAX: mathematical programming in JAX // ArXiv Preprint ArXiv:2412.09734. 2024. <https://arxiv.org/abs/2412.09734>

26. *Ryu E., Chen Y., Li W., Osher S.* Vector and matrix optimal mass transport: theory, algorithm, and applications // *SIAM J. Scientif. Comput.* 2018. V. 40. No. 5. P. A3675–A3698.
27. *Degleris A., Gamal A., Rajagopal R.* GPU accelerated security constrained optimal power flow // *ArXiv Preprint ArXiv:2410.17203*. 2024.  
<https://arxiv.org/abs/2410.17203>
28. *Ryu M., Byeon G., Kim K.* A GPU-accelerated distributed algorithm for optimal power flow in distribution systems // *ArXiv Preprint ArXiv:2501.08293*. 2025.
29. *Wang X., Zhang Q., Ren J., Xu S., Wang S., Yu S.* Toward efficient parallel routing optimization for large-scale SDN networks using GPGPU // *Journal Network Comput. Appl.* 2018. V. 113. P. 1–13.
30. *Kikuta K., Oki E., Yamanaka N., Togawa N., Nakazato H.* Effective parallel algorithm for GPGPU-accelerated explicit routing optimization // *2015 IEEE Global Communications Conference. IEEE*, 2015. P. 1–6.
31. *Zhang S., Ajayi O., Cheng Y.* A self-supervised learning approach for accelerating wireless network optimization // *IEEE Transactions on Vehicular Technology*. 2023. V. 72. No. 6. P. 8074–8087.
32. *Wu J., He Z., Hong B.* Chapter 5 – Efficient CUDA algorithms for the maximum network flow problem // *GPU Computing Gems Jade Edition*. Boston: Morgan Kaufmann, 2012. P. 55–66.  
<https://www.sciencedirect.com/science/article/pii/B9780123859631000058>
33. *Liu H., Huang S., Qin S., Yang T., Yang T., Xiang Q., Liu X.* Keep your paths free: Toward scalable learning-based traffic engineering // *Proceedings of the 8th Asia-Pacific Workshop on Networking*. 2024. P. 189–191.  
<https://doi.org/10.1145/3663408.3665813>
34. Gurobi Optimization. LLC Gurobi Optimizer Reference Manual. 2024.  
<https://www.gurobi.com>
35. *Yarmoshik D., Pershianov M.* On the application of saddle-point methods for combined equilibrium transportation models // *Proceedings of the 23rd International Conference on Mathematical Optimization Theory and Operations Research (MOTOR)*. 2024. P. 432–448. [https://doi.org/10.1007/978-3-031-62792-7\\_29](https://doi.org/10.1007/978-3-031-62792-7_29)
36. *Kubentayeva M., Yarmoshik D., Pershianov M., Kroshnin A., Kotliarova E., Tupitsa N., Pasechnyuk D., Gasnikov A., Shvetsov V., Baryshev L., Shurupov A.* Primal-dual gradient methods for searching network equilibria in combined models with nested choice structure and capacity constraints // *ArXiv Preprint ArXiv:2307.00427*. 2023. <https://arxiv.org/abs/2307.00427>
37. *Malitsky Y., Pock T.* A first-order primal-dual algorithm with linesearch // *SIAM J. Optim.* 2018. V. 28. No. 1. P. 411–432.
38. *Zhu M., Chan T.* An efficient primal-dual hybrid gradient algorithm for total variation image restoration // *UCLA CAM Report*. 2008. V. 34. No. 2.
39. *Chambolle A., Pock T.* A first-order primal-dual algorithm for convex problems with applications to imaging // *J. Math. Imaging Vision*. 2011. V. 40. P. 120–145.
40. *Chambolle A., Pock T.* On the ergodic convergence rates of a first-order primal-dual algorithm // *Mathematical Programming*. 2015. V. 159. P. 253–287.
41. *Parikh N., Boyd S.* Proximal algorithms // *Foundations and Trends in Optimization*. 2014. V. 1. No. 3. P. 127–239.

42. *Held M., Wolfe P., Crowder H.* Validation of subgradient optimization // Mathematical Programming. 1974. V. 6. P. 62–88.
43. *Duchi J., Shalev-Shwartz S., Singer Y., Chandra T.* Efficient projections onto the  $l_1$ -ball for learning in high dimensions // Proceedings of the 25th International Conference on Machine Learning. 2008. P. 272–279.  
<https://doi.org/10.1145/1390156.1390191>
44. *Condat L.* Fast projection onto the simplex and the  $l_1$  ball // Mathematical Programming. 2016. V. 158. No. 1-2. P. 575–585.  
<https://doi.org/10.1007/s10107-015-0946-6>

*Статья представлена к публикации членом редколлегии П.С. Щербаковым.*

Поступила в редакцию 03.03.2025

После доработки 20.05.2025

Принята к публикации 27.06.2025