# Price-Directive Methods in Multicommodity Flow Problem

**Yihang Chen**
Department of Mathematics
Peking University
1700010780

**Yunchang Yang**
Center for Big Data Science
Peking University
1901111594

## 1 Introduction

The multicommodity network flow (MCNF) problem seeks optimal flow assignment of different commodities to be shipped between specified origins and destinations, where flows of different commodities compete for the same arc capacities. It has very broad applications in the management of telecommunication, transportation, and logistics networks.

The MCNF can model a wide variety of real-world problem, and a lot of case specific variants exist. We can split them in three main categories depending on the domain of the decision variables, either they are integers, then the problem is called integer MCNF, approximation algorithm are mainly used in this case. Second, the decision variables take on real values and the problem can be modeled as a LP. The last category is mixed integer MCNF, where some decision variables are integers and others real values. It has been proved that in integer and mixed integer MCNF, the problem becomes NP-hard.

Today most of the methods employed to solve the MCNF use "decomposition" techniques and can be classified in three categories:

1. **Price-directive Methods**:
   Price-directive methods are based on the idea that by associating the bundle constraints with ”correct” penalty functions (dual prices, or Lagrangian multipliers), a hard MCNF problem can be decomposed into k easy SCNF problems. Some widely use price-directive methods are Lagrangian relaxation or Dantzig-Wolfe decomposition.
2. **Resource-directive Methods**:
   These methods aim for reducing the MCNF to multiple single commodity flow problem which can then be solved very efficiently. To do so, a new constraint is introduced which allocate a certain amount of capacity on each arc for each commodity, without exceeding the total original arc capacity. However these methods does not ensure convergence.
3. **Partitioning methods**:
   Most of the well-known LP solver methods can be adjusted to the MCNF specifically, for instance the "network simplex" is a speed-up modified version of simplex method.

It has been shown that the resource-directive algorithms converge rapidly for small problems but are outperformed by the price-directive techniques for large instances.

While the classical multicommodity flow problem tries to minimize the total cost of the network flow, we focus our attention on the link capacity. We want the network load to be balanced across links so as to avoid network congestions and we consider to minimize the maximum (or worst-case) link utilization ratio to achieve load balancing. Our model is based on the path-based formulation of the multicommodity flow framework.

## 2 Problem Formulation

### 2.1 Generic Definition

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed network, where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and uni-directional edges with sizes $N$ and $L$ respectively. For each edge $l$ ($l = 1, 2, \cdots, L$), the link capacity is denoted as $c_l \in \mathbb{R}^+$, and $\mathbf{c} = (c_1, c_2, \cdots, c_L)^\top$ to be the link capacity vector of the overall network.

The communication network is used to deliver data flows from source nodes to destination nodes to facilitate end to-end data services. Specifically, we consider $K$ flows in the network indexed by $k (k = 1, 2, \cdots, K)$. For each flow $k$, there is a source-destination pair associated with it and we assume that there are $P_k \in \mathbf{Z}^+$ available paths for this source-destination pair indexed by $p_k$ ($p_k = 1, 2, \cdots, P_k$).

We use a $L \times P_k$ routing matrix to represent the relationship between the links and the available paths of flow $k$, $\mathbf{R}_k = (R^k_{l,p_k})$, where $R^k_{l,p_k} = \{0, 1\}$. $R^k_{l,p_k} = 1$ if and only if the path $p_k$ transverses link $l$ for flow $k$.

Let $\mathbf{x}_k = (x_{k,1}, x_{k,2}, \cdots, x_{k,P_k})$ be the rate allocation and path selection vector of flow $k$, where each element $x_{k,p}(x_{k,p} \geq 0, \forall k, p)$ measures the rate allocation at path $p$ for flow $k$. $x_{k,p}$ also gives information about what paths to select for flow $k$. Specifically, the path selected by flow $k$ are those paths with $x_{k,p_k} > 0$ for any $p_k$. Define $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, x_K)$ to be the rate allocation and path selection vector of all flows.

Our destination is to balance the network load across links, which can be formulated by an optimization problem

$$\begin{array}{ll} \min_{\mathbf{x}} & \max_l \frac{\mathbf{R}[l]\mathbf{x}}{c_l} \\ \text{s.t.} & \mathbf{R}\mathbf{x} \leq \mathbf{c} \\ & \|\mathbf{x}_k\|_1 = d_k \\ & \mathbf{x} \geq 0 \end{array} \tag{1}$$

By introducing an additional variable $t$, we can transform the nonlinear optimization problem into the following LP:

$$\min_{\mathbf{x},t} \quad t \tag{2a}$$

$$\text{s.t.} \quad \mathbf{R}\mathbf{x} \leq \mathbf{c}t \tag{2b}$$

$$1^\top \mathbf{x}_k = d_k \tag{2c}$$

$$t \leq 1 \tag{2d}$$

$$\mathbf{x} \geq 0 \tag{2e}$$

## 3 Classical MCNF

The problem we deal with in our work has a very close relationship with the classical Multicommodity Network Flow (MCNF) problem. Here we provide a brief description. Note that some definitions, such as link cost and node capacity, are not included in our algorithm.

### 3.1 Arc-Node Model

Given a directed graph $G := (N, A)$, the arcs are defined by their start and end nodes $(i, j)$, and their cost per unit $c_{ij}$. Each node $n \in N$ and arc $(i, j) \in A$ have a maximum capacity $u_n$ and $v_{ij}$ respectively. Moreover, a commodities $k \in K$ is characterized by its start and end nodes, as well as a quantity $q_k$. Finally, to simplify the expression of the flow conservation constraints, we define a variable $\delta^k_n$ which equals 1 if the node $n$ is the start node of the commodity $k$, equals -1 if $n$ is the end node, otherwise it equals 0. The decision variable $x^k_{ij}$ correspond to the fraction of the total quantity of the commodity $k$ passing through the arc $(i, j)$.

The node-arc formulation is defined as follow:

$$\begin{array}{ll} \text{minimize} & \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}x_{ij}^k q_k \\ \text{subject to} & \sum_{i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \delta_i^k \quad \forall i \in N, \forall k \in K \\ & \sum_{k\in K}\sum_{(j,i)\in A} x_{ji}^k q_k \leq u_i \quad \forall i \in N \\ & \sum_{k\in K} x_{ij}^k q_k \leq v_{ij} \quad \forall (i,j) \in A \\ & x_{ij}^k \geq 0 \quad \forall (i,j) \in A, \forall k \in K \end{array} \tag{3}$$

This formulation aims to minimize the total cost of shipping all the commodities, while conserving, for each commodity, the entire flow between the start and end nodes. Moreover, the total quantity of the commodities at each node and arcs is limited, and the commodities are allowed to split during the shipping.

### 3.2 Path-Based Model

Another widely used MCNF formulation is called path-based model. Given a graph $G := (N, A)$ similar to the one defined in the arc-node model, nodes and arcs have a capacity $u_n$ and $v_{ij}$ respectively, the cost per unit of an arc is $c_{ij}$, and the quantity of the commodity $k$ is $q_k$. Also, we define the sets $P(k)$ such that it contains all the possible paths from the start and end node of the commodity $k$. The cost per unit of a path $p$ is denoted $c_p$, it corresponds to the sum of $c_{ij}$, for all arcs $(i, j)$ contained in the path $p$. Unlike the arc-node variant, the decisions variables are $x_p^k$ which is the fraction of the total quantity of the commodity $k$ passing through the path $p \in P(k)$ Besides, to simplify the formulation of the capacity constraints, we define $\alpha_{ij}^p$ to be equals to 1 if the arc $(i, j)$ is a part of the path $p$. As well as $\beta_n^p$ to be equal to 1 if the node $n$ is contained in the path $p$ and is not the origin of the path $p$.

The path-based formulation is described as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{k\in K}\sum_{p\in P(k)} c_p x_p^k q_k \\ \text{subject to} & \sum_{p\in P(k)} x_p^k = 1 \quad \forall k \in K \\ & \sum_{k\in K}\sum_{p\in P(k)} x_p^k q_k \beta_i^p \leq u_i \quad \forall i \in N \\ & \sum_{k\in K}\sum_{p\in P(k)} x_p^k q_k \alpha_{ij}^p \leq v_{ij} \quad \forall (i,j) \in A \\ & x_p^k \geq 0 \quad \forall p \in P(k), \forall k \in K \end{array} \tag{4}$$

Similarly, the objective is to minimize the total cost of transporting all the commodities, while conserving, for each commodity all the flow from the start node to the end node. Besides the nodes and arcs have maximal capacities and the commodities can split during the transportation.

This formulation allow us to visualize the MCNF in another way, instead of focusing on the arcs, we look at the bigger picture and compute how good it is to assign a certain amount of a commodity to a path. Our problem is like the path-based model.

### 3.3 Comparison of different models

Thus, we defined the MCNF in two different ways. It would be interesting to know in which cases a formulation is more efficient than the other one.

First, observe that in general, the number of variables in the path-based formulation is highly bigger than in the arc-node model. Besides the number of variable grows exponentially with the number of arcs in the path-based model. However, notice that the number of flow conservation constraints is highly lower in the path-based model. As the number of commodities grows, the number of constraints in the arc-node model grows greatly faster than in the path-based model.

Previous literatures show that for the small dataset the running time of the arc-node formulation is extremely faster than the path-based model. Indeed, the later require a large amount of precomputational time, to generate the set $P(k)$ for each commodity and the number of variables is larger in the path-based formulation. Besides, the path based model requires much more memory than the arc-node formulation, again it can be explained by the large difference of number of variables generated between the two models.

However, we notice that the main benefit of the path-based approach is the low number of constraints. Moreover, most of the time the optimal solution contains only a small amount of variables among all

the ones generated, meaning that we do not need all variables to obtain the optimal solution. One way to leverage these observations would be to employ a method which does not require all the variables to find the optimal solution, but only the relevant ones. From the literature review we have seen that a wide variety of approaches exist, with the aim of cutting down the amount of variables considered. One of them is the column generation approach, which is today extensively used.

### 3.4 Column Generation on LP

Column generation is a method to efficiently solve linear programs with a huge number of variables. The principal idea works as follows: We want to solve an LP, called master LP (MP), and consider a restricted master LP (RMP), which contains all constraints of the master LP, but only a subset of the variables. Then we solve a pricing problem, i.e., we decide whether there is a variable that is currently not contained in the restricted master LP, but might improve the objective function. If there are no such variables, it is guaranteed that the current solution of the restricted master LP is optimal for the whole problem. Otherwise, we add the variables and iterate.

Consider the following linear program, the so-called master problem (MP):

$$\begin{aligned} \max \quad & \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{MP}$$

where $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. A vector $x \in \mathbb{R}^n$ is feasible for the primal program if $A\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \geq \boldsymbol{0}$. The dual program is

$$\begin{aligned} \min \quad & b^{\mathrm{T}}y \\ \text{s.t.} \quad & A^{\mathrm{T}}y \geq c \\ & y \geq 0 \end{aligned} \tag{D-MP}$$

We have the following duality theorem: $x^*$ and $y^*$ are respectively the optimal solution of the primal and dual problem if and only if the three following conditions hold :

- Primal feasibility : $x^*$ satisfies each constraint of primal problem.
- Objective function : The value of the primal objective function for $x^*$ is the same as the dual objective function value for $y^*$.
- Dual feasibility : $y^*$ satisfies each constraint of dual problem.

Now consider the following RMP with respect to a certain subset of variables $\{x_{p'} : p' \in P' \subset [n]\}$:

$$\begin{aligned} \max \quad & \sum_{p'} c_{p'} x_{p'} \\ \text{s.t.} \quad & \sum_{p'} A_{mp'} x_{p'} \leq b_m \quad \forall m \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{RMP}$$

A feasible solution $(x'_p)_{p \in P'}$ for (RMP) yields a feasible solution $(x_p)_{p \in [n]}$ for (MP) by setting $x_p = x'_p$ for $p \in P'$, and $x_p = 0$ otherwise. It follows that the optimal solution value $v$(RMP) of (RMP) is not larger than the optimal value of (MP). We want to decide whether the solution is optimal for (MP). If this is not the case, we want to add additional variables to (RMP) that help improve the solution value. How can we do that?

Recall the duality theorem, given the optimal solution to the RMP called $(x'_p)_{p \in P'}$ and its dual variables $(y'_p)$ we can confirm that $(x'_p)_{p \in P'}$ satisfies the Primal feasibility condition as the RMP contains the same constraints as the MP. Besides the Objective function condition is also satisfied thanks to the strong duality theorem. The only condition left that we must verify, to ensure that the optimal solution of the RMP is also the optimal solution to the MP, is the dual feasibility. That is, verify that $(y'_p)$ satisfy all the constraints of the dual of the MP, which is called the pricing problem.

If the dual variables of (RMP) satisfies the restrictions of (D-MP), then from the duality theorem we are done. If there exists a variable $p$ for which D-MP is violated, we add the corresponding variable to (RMP) and repeat the process.

Let $y_k$ with $k \in K, y_{(i)}$ with $i \in N$ and $y_{(i,j)}$ such that $(i,j) \in A$ denote the dual variables corresponding to the $k^{th}$ constraint of the flow constraints, the $i^{th}$ constraint of the node constraints, and to the $(i,j)^{th}$ constraint of the arc constraints in (4) respectively. The dual problem of the MP is formulated as follows:

$$\begin{array}{lll} \text{maximize} & \sum_{k\in K} y_k + \sum_{i\in N} y_{(i)} u_i + \sum_{(i,j)\in A} y_{(i,j)} v_{ij} & \\ \text{subject to} & y_k + \sum_{i\in N} y_{(i)} q_k \beta_i^p + \sum_{(i,j)\in A} y_{(i,j)} q_k \alpha_{ij}^p \leq c_p^k q_k & \forall k \in K, \forall p \in P(k) \\ & y_k \in \mathbb{R} & \forall k \in K \\ & y_{(i)} \leq 0 & \forall i \in N \\ & y_{(i,j)} \leq 0 & \forall (i,j) \in A \end{array} \tag{5}$$

The aim is to verify whether the dual variable satisfies all the constraints, this problem is called the Pricing problem. Nonetheless, the amount of constraints is equal to the number of variables in the MP, which is extremely large. This huge quantity of variables was the reason why we needed to find a new efficient approach to exploit the path-based model. Thus, the key is to identify an efficient technique to verify all those constraints without explicitly enumerate all of them.

We are going to formulate the constraints in another way to tackle the pricing problem from a different point of view :

$$\begin{aligned} \forall k \in K, \forall p \in P(k): \quad & y_k + \sum_{i\in N} y_{(i)} q_k \beta_i^p + \sum_{(i,j)\in A} y_{(i,j)} q_k \alpha_{ij}^p \leq c_p^k q_k \\ & \equiv -y_k - \sum_{i\in N} y_{(i)} q_k \beta_i^p - \sum_{(i,j)\in A} y_{(i,j)} q_k \alpha_{ij}^p + c_p^k q_k \geq 0 \\ & \equiv -y_k - \sum_{(i,j)\in A} y_{(j)} q_k \alpha_{ij}^p - \sum_{(i,j)\in A} y_{(i,j)} q_k \alpha_{ij}^p + q_k \sum_{(i,j)\in A} c_{ij}^k \alpha_{ij}^p \geq 0 \\ & \equiv -y_k + q_k \sum_{(i,j)\in A} \alpha_{ij}^p \left(c_{ij}^k - y_{(j)} - y_{(i,j)}\right) \geq 0 \end{aligned} \tag{6}$$

Consequently, for each commodity $k \in K$ all of the following inequalities need to be satisfied:

$$-y_k + q_k \sum_{(i,j)\in A} \alpha_{ij}^p \left(c_{ij}^k - y_{(j)} - y_{(i,j)}\right) \geq 0 \quad \text{subject to } : p \in P(k) \tag{7}$$

Which is equivalent to the system:

$$-y_k + q_k \sum_{(i,j)\in A} \alpha_{ij}^p \left(c_{ij}^k - y_{(j)} - y_{(i,j)}\right) \geq 0 \tag{8}$$

$$\text{subject to :} \quad \sum_{(i,j)\in A} \alpha_{ij}^p - \sum_{(j,i)\in A} \alpha_{ji}^p = \gamma_i^k \tag{9}$$

$$\alpha_{ij}^p \in \{0,1\} \quad \forall i \in N \tag{10}$$

where $\gamma_i^k$ equals 1 if $i$ is the starting node of the commodity $k$, equals -1 if $i$ is the destination of $k$ and 0 otherwise. Notice that $P$ denotes all paths in the network.

Finally, we can rewrite the pricing problem as Integer Programs, the following problem will be called sub-pricing problem for commodity $k$:

$$\begin{array}{lll} \text{minimize} & -y_k + q_k \sum_{(i,j)\in A} \alpha_{ij} \left(c_{ij}^k - y_{(j)} - y_{(i,j)}\right) & \\ \text{subject to :} & \sum_{(i,j)\in A} \alpha_{ij} - \sum_{(j,i)\in A} \alpha_{ji} = \gamma_i^k & \forall i \in N \\ & \alpha_{ij} \in \{0,1\} & \forall (i,j) \in A \end{array} \tag{11}$$

Indeed if the objective function value of the optimal solution is negative, then it implies that the path $p$ described by the arcs $\alpha_{ij}$ 's violates a constraint in the dual of the MP. Thus the corresponding variable

in the RMP needs to be generated to satisfy the constraint, in order to ensure that the constraint will be satisfied later on. Finally, if for each commodity, the objective function value of the optimal solution is nonnegative, it implies that the corresponding $\mathrm{RMP}$ solution is also optimal to the MP.

The constraints aim to produce a set of arcs such that it contains exactly one arc starting from the origin of the commodity $\mathrm{k}$, and one arc ending at the destination of commodity $\mathrm{k}$, moreover the set of arcs must form a path. Thus, the feasible $\alpha_{ij}$ 's vectors represent all the paths from the origin of commodity $k$ to its destination. On the other hand, the objective function contains one constant $y_k$, and the other term is a cost vector modified by the dual variables $y_{(i)}$ and $y_{(i,j)}$. Therefore, this integer program is in fact a shortest path problem with penalized arcs costs. Hence, the pricing problem boils down to a set of $\|K\|$ shortest path problems.

### 3.5 Initial Paths Generation

The first task is to find a method to find a promising set of initial variables. We use the Dijkstra's algorithm to compute the shortest path between one origin-destination pair. However, owing to the link capacity, some additional modifications must be made.

For a given OD-pair $i$, we first compute a path linking these two nodes. In sequel, we check whether the commodity carried by this path satisfies the link capacity. Assume the link can be denoted by $a_0 a_1 \cdots a_n$. For $i = n, n-1, \cdots, 1$, we check whether the link $a_{i-1}a_i$ satisfies the capacity constraint. After routing over the whole path, if necessary, we decrease the amount of the commodities carried by this path according to each capacity constraint.

Finally, we update the link capacity, i.e. subtracts the amount that has been occupied by this path. If one link's capacity is reduced to zero, we choose to ignore it in the future Dijkstra's algorithm.

Clearly, it may occur that one origin-destination pair might be unavailable because of the link reduction. To prevent this from happening, we adopt the following strategy:

1. For the fist epoch, we run the Dijkstra's algorithm directly to compute the shortest path.
2. The commodities are sorted according to the outdegree value of their shortest path, which equals the sum of the outdegree of each nodes the path contains, except the destination node. The intuition is that the outdegree of a path tend to represent the number of alternative path from two nodes.
3. Iterate over the commodities according to the order. If again, one OD-pair is infeasible, we rerun the algorithm and put this OD-pair in the beginning.

For most cases, such algorithm is able to ensure the feasible initial paths.

## 4 Lagrange Relaxation

### 4.1 General Description

Lagrangian relaxation dualizes the bundle constraints using a Lagrangian multiplier $\pi \geq 0$ so that the remaining problem can be decomposed into $K$ smaller min-cost network flow problems. In the literature it is widely used for node-arc formulations. Below we describe its general form.

Suppose that we want to solve the following optimization problem:

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & Ax = b \end{array} \tag{12}$$

The Lagrange dual function is

$$L(x, z) = f(x) + z^\top (Ax - b) \tag{13}$$

And solving the primal problem is equivalent to soling the following max-min problem:

$$\max_z \min_x L(x, z) \tag{14}$$

which can be solved using the following scheme, called dual gradient ascent method:

$$\begin{aligned} x^{k+1} &= \operatorname{argmin}_x L\left(x, z^k\right) \\ &= \operatorname{argmin}_x \left(f(x) + \left(z^k\right)^T Ax\right) \\ z^{k+1} &= \operatorname{argmax}_z L\left(x^{k+1}, z\right) - \frac{1}{2t}\left\|z - z^k\right\|_2^2 \\ &= z^k + t\left(Ax^{k+1} - b\right) \end{aligned} \tag{15}$$

### 4.2 Application to our problem

In particular, we apply the Lagrange relaxation method to our multicommodity flow problem. In the $i$-th iteration, We introduce dual variables $\lambda^i \in \mathbb{R}^L$ for the capacity constraints, and the relaxed primal problem can be written as

$$\min_{\mathbf{x},t} \quad t + \lambda^\top (\mathbf{R}\mathbf{x} - \mathbf{c}t) \tag{16a}$$
$$\text{s.t.} \quad 1^\top \mathbf{x}_k = d_k \tag{16b}$$
$$\mathbf{x} \geq 0 \tag{16c}$$
$$0 \leq t \leq 1 \tag{16d}$$

this problem can be further decomposed into $K+1$ smaller problems: for each $k \in \{1, ..., K\}$,

$$\min_{\mathbf{x}_k} \quad \lambda^\top (\mathbf{R}_k \mathbf{x}_\mathbf{k}) \tag{17a}$$
$$\text{s.t.} \quad 1^\top \mathbf{x}_k = d_k \tag{17b}$$
$$\mathbf{x} \geq 0 \tag{17c}$$

and

$$\min_{t} \quad t + \lambda^\top (-\mathbf{c}t) \tag{18a}$$
$$\text{s.t.} \quad 0 \leq t \leq 1 \tag{18b}$$

For the $k$-th subproblem, we can view it as a min-cost problem, with cost of the $l$-th arc $\lambda_l^i$ . We then use the column generation method mentioned in Section 3. We first solve the restricted master problem:

$$\min_{\mathbf{x}_k} \quad \lambda^\top \left(\mathbf{R}_k^B \mathbf{x}_\mathbf{k}\right) \tag{19a}$$
$$\text{s.t.} \quad 1^\top \mathbf{x}_k = d_k \tag{19b}$$
$$\mathbf{x} \geq 0 \tag{19c}$$

where $\mathbf{R}_k^B$ contains a subset of all columns (paths) of $\mathbf{R}_k$. And we generate the corresponding dual variable. Then, we aim to verify whether the all the dual constraints are satisfied, which are of huge amount. Hence, we might formulate it as a pricing problem, and use the method mentioned in Section 3 to solve it. And we repeat the process until there is no dual constraint violated.

After we solve the subproblems, we update $\lambda^i$ using subgradient method:

$$\lambda^{i+1} = \lambda^i + \alpha_i (\mathbf{R}\mathbf{x} - \mathbf{c}t) \tag{20}$$

Subgradient methods are common techniques for determining the Lagrangian multipliers. They are easy to implement but have slow convergence rates. They usually do not have a good stopping criterion, and in practice usually terminate when a predefined number of iterations or nonimproving steps is reached. In the experiment we set $\alpha_i = 1/i$.

# 5 Dantzig-Wolfe Decomposition

## 5.1 Algorithm Description

This part of the report gives a detailed description of the Dantzig Wolfe Decomposition [1] in solving the LP of block angular form. The notations are used solely in this part of the report.

Consider a linear program of the form

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b, \quad x \ge 0 \end{aligned} \tag{21}$$

We can define the constraint matrix is of the form

$$A = \begin{bmatrix} L_1 & L_2 & \cdots & L_K \\ A_1 & & & \\ & A_2 & & \vdots \\ & & \ddots & \\ & & & A_K \end{bmatrix} \tag{22}$$

Such a form for $A$ is called block angular, and the linear system is defined by

$$\begin{aligned} \min \quad & \sum_{k=1}^{K} (c^k)^\top x^k \\ \text{s.t.} \quad & \sum_{k=1}^{L} L_k x^k \le b^0 \\ & A_k x^k \le b^k \quad k = 1, 2, \cdots, K \\ & x^k \ge 0 \quad k = 1, 2, \cdots, K \end{aligned} \tag{23}$$

It is not enough to have the decomposition into master and subproblems as above. To make the decomposition effective, the master problem needs reformulation. The reformulation will enable the master problem and subproblems to exchange information regarding progress toward an optimal solution for the original linear program while enabling each problem to be solved separately. The key to reformulation is the fact that the subproblems are all linear pro- grams and so the feasible sets are polyhedrons.

Let $P_k = \{x^k | A_k x^k \le b^k\}$ be the extreme feasible sets of the constraints and $v_1^k, v_2^k, \cdots, v_{N_k}^k$ be the extreme points of $P_k$ and $d_1^k, d_2^k, \cdots, d_{l_k}^k$ be the extreme directions of Pk. Then, any point $x^k \in P_k$ can be expressed as

$$x^k = \sum_{i=1}^{N_k} \lambda_i^k v_i^k + \sum_{j=1}^{l_k} \mu_j d_j^k \tag{24}$$

where $\sum_{i=1}^{N_k} \lambda_i^k = 1, \lambda_i^k \ge 0, \mu_j^k \ge 0$.

The the Dantzig-Wolfe decomposition of the original LP is (23) is

$$\begin{aligned} \min \quad & \sum_{i=1}^{N_k} \lambda_i^k p_i^k + \sum_{j=1}^{l_k} \mu_j^k \bar{p}_j^k \\ \text{s.t.} \quad & \sum_{k=1}^{K} (\sum_{i=1}^{N_k} \lambda_i^k q_i^k + \sum_{j=1}^{l_k} \mu_j^k \bar{q}_j^k) \le b^0 \\ & \sum_{i=1}^{N_k} \lambda_i^k = 1 \\ & \lambda_i^k, \mu_j^k \ge 0 \end{aligned} \tag{25}$$

where $p_i = (c^k)^\top v_i^k, \bar{p}_j^k = (c^k)^\top d_j^k, q_i^k = L_k v_i^k$ and $\bar{q}_j^k = L_k d_j^k$. The constraint $\sum_{k=1}^K (\sum_{i=1}^{N_k} \lambda_i^k q_i^k + \sum_{j=1}^{l_k} \mu_j^k \bar{q}_j^k)$ is called the linking constraint, with dual variable denoted as $\pi^1$, and $K$ constraint $\sum_{i=1}^{N_k} \lambda_i^k = 1$ are called convexity constraint, with dual variable denoted as $\pi_k^2$.

### 5.2 Column Generation on the Dantzig Wolfe Decomposition

The number of variables $\lambda_i^k$ and $\mu_j^k$ can be extremely large for even moderately sized problems since feasible sets of subproblems (linear programs) can have an extremely large number of extreme points and extreme directions. In other words, there will be many more columns than rows in the reformulated master problem.

Actually, the node-arc formulation of the classical multicommodity problem corresponds to an LP with the block angular form in (23). After the reformulation, it has been shown that it is equivalent to the path-based formulation of the multicommodity problem. Since our problem has been path-based, the above procedures are not necessary. We will focus on the following procedures of the Dantzig-Wolfe decomposition, in other words, the column generation methods.

The key idea to get around this difficulty of handling an extremely large number of variables is to use the revised simplex method to solve the master problem. The major advantage is that it is not necessary to formulate the entire reformulated master problem since the vast majority of the variables will be zero (i.e., non-basic) at an optimal (basic feasible) solution. This motivates the construction of a smaller version of the master problem called the restricted master problem where only a small subset of the variables $\lambda_i^k$ and $\mu_j^k$ included corresponding to a current basic feasible solution, and the remaining variables are non-basic, i.e., set to zero. If the reduced costs of the basic feasible solution are non-negative, then the revised simplex method stops with the optimal solution, else some non-basic variable with negative reduced cost is selected to enter the basis.

Given a current basis $B$ for the restricted master problem, solve the liear system $B^\top \pi = q_B$, where $q_B$ is a vector consisting of the quantities $q_i^k$ and $\bar{q}_i^k$ associated with variables $\lambda_i^k$ and $\mu_j^k$, which are basic.

We assume that the components of $\pi$ are arranged so that $\pi^\top = [(\pi^1)^\top, \pi_1^2, \cdots, \pi_K^2]$. Then the reduced cost corresponding to a non-basic variable $\lambda_i^k$ is of the form

$$r_i^k = p_i^k - \pi^\top \begin{bmatrix} q_i^k \\ e_k \end{bmatrix} \tag{26}$$

and the reduced cost for corresponding to a non-basic variable $\mu_j^k$ is of the form

$$\bar{r}_j^l = \bar{p}_j^k - \pi^\top \begin{bmatrix} \bar{q}_i^k \\ 0 \end{bmatrix} \tag{27}$$

the main difference comes from the convexity constraint.

Since we want to minimize all the $r_i^k$, we have

$$r_{\min} = \min_k \left\{ \min_i (c^k)^\top v_i^k - (\pi^1)^\top L_k v_i^k - \pi_k^2 \right\} \tag{28}$$

Let $r_\star^k = \min_i r_i^k$, where $r_i^k$ is equivalent to the optimal objective function of the subproblem

$$\begin{aligned} \min \quad & ((c^k)^\top - (\pi^1)^\top L_k) x^k - \pi_k^2 \\ \text{s.t.} \quad & A_k x^k \le b^k, \quad x^k \ge 0 \end{aligned} \tag{29}$$

since $v_i^k$s are the extreme points of the polygon $A_k x^k \le b^k$.

There are three possibilities in solving the subproblems in attempting to generate $r_{\min}$.

1. If all subproblems are bounded and $r_{\min} < 0$, then let $t$ be the index of minimal value. The column associated with the optimal extreme point $\lambda_i^t$ is added into the basis $B$.
2. If all subproblems are bounded and $r_{\min} \ge 0$, then the current basis $B$ is optimal.
3. If there is at least one subproblem that is unbounded, then let $s$ be the index $k$ of such an unbounded subproblem. The revised simplex method will return an extreme direction $d_j^s$ associated with the subproblem such that $((c^s)^\top - (\pi^1)^\top L_s) d_j^s < 0$ and so the column associated with $\mu_j^s$ can enter the basis $B$.

### 5.3 Application to the Muticommodity Problem

In this part, we focus on applying the Dantzig-Wolfe decomposition to our multicommodity problem. The notations are adopted from Section (2.1).

For the master problem 1, we introduce the dual variable $\pi^1 \geq 0$ for (2b), $\sigma_k, 1 \leq k \leq K$ for (2c), and $u$ for the equation 2d. We can write the Lagrangian to be

$$L(\mathbf{x}, t, \pi, \sigma, u) = t + \pi^\top(\mathbf{R}\mathbf{x} - ct) + \sum_{k=1}^{K} \sigma_k(d_k - 1^\top x_k) + u(t-1) \tag{30}$$

Hence, the dual problem can be written as (deleting $u$)

$$\begin{aligned} \min_{\pi,\sigma} \quad & c^\top \pi - \sigma^\top d \\ \text{s.t.} \quad & R_k^\top \pi \geq \sigma_k 1 \\ & c^\top \pi \geq 1 \\ & \pi \geq 0 \end{aligned} \tag{31}$$

For the Master Problem (1), we first solves a Restricted Master Problem:

$$\min_{\mathbf{x},t} \quad t \tag{32a}$$
$$\text{s.t.} \quad \mathbf{R_B}\mathbf{x} \leq \mathbf{c}t \tag{32b}$$
$$1^\top \mathbf{x}_k = d_k \tag{32c}$$
$$t \leq 1 \tag{32d}$$
$$\mathbf{x} \geq 0 \tag{32e}$$

where $\mathbf{B}$ is a subset of all columns (paths) of $\mathbf{R}$, and generate the corresponding dual variable. Then, we aim to verify whether the all the dual constraints are satisfied, which are of huge amount. Hence, we might formulate it as a pricing problem.

Since $\mathbf{R}$ encode the information of the path and the link, we will decompose the relation in the following. For the commodity $k$, and any path $p_k \in P(k)$, we want to verify whether $(R_k^p)^\top \pi \geq \sigma_k$ holds. In other words, whether $\min_{p_k} -\sigma_k + \sum_l R^k_{l,p_k}\pi_l \geq 0$.

If we denote the link $l$ as link $(i,j)$, where $i, j$ represent the nodes. Any path $p_k$ must obeys the relation:

$$\sum_{(i,j)\in\mathcal{E}} R^k_{(i,j),p} - \sum_{(j,i)\in\mathcal{E}} R^k_{(j,i),p} = \alpha_i^k. \tag{33}$$

where $\alpha_i^k$ equals 1 if $i$ is the starting node of the commodity $k$, equals $-1$ if i is the destination of $k$ and 0 otherwise.

Finally, we can rewrite the pricing problem as Integer Programs. For each commodity $k$, we solve the following integer program:

$$\begin{aligned} \min_{R} \quad & -\sigma_k + \sum_l R^k_{l,p}\pi_l \\ \text{s.t.} \quad & \sum_{(i,j)\in\mathcal{E}} R^k_{(i,j),p} - \sum_{(j,i)\in\mathcal{E}} R^k_{(j,i),p} = \alpha_i^k \\ & R^k_{l,p} \in \{0,1\} \end{aligned} \tag{34}$$

Indeed if the objective function value of the optimal solution is negative, then it implies that the path $p$ obtained via solving the Integer Problem violates a constraints in the dual of the Master Problem. This integer problem can also be solved by the famous Dijkstra's algorithm. We need only regard $\pi_l$ as cost on each edge.

# 6 Dataset Preprocessing

## 6.1 Input Format

We adopt the input format suggested in [2], where three types of formulations have been discussed.

1. **Origin-Destination Problem (ODP)**
   We define a commodity as a product that travels between a specific origin and a specific destination.
2. **Destination Specific Problem (DSP)**
   We define a commodity as a product that travels to a specific destination from multiple origins, or vice versa, from a specific origin to multiple destinations.
3. **Product Specific Problem (PSP)**
   We define a commodity as a product that must travel through a network from multiple origins to multiple destinations.

The data will be split into 4 sections: a node file for general information about the network, a link file, a supply/demand file with the name of the following 4 files, xxx.nod, xxx.arc, xxx.sup and xxx.mut.

Notice that our problem belongs to the Origin-Destination Formulation, and xxx.sup cannot be directly applied to the ODP, since some origin or destination of a given flow is not specified (denoted as "-1"). Some datasets include an additional file xxx.od. Most instances of the JLF problems [2] have the Origin-Destination Formulation, hence they have both a ".sup" file for the Product-Specific (aggregated) Formulation and a ".od" file for the OS (disaggregated) formulation.

## 6.2 Data Input

The dataset contains 4 files: .nod, .arc, .od, .mut.

We read the data as the following order:

- First, we read .nod file. This file contains 4 numbers, of which 2 numbers are useful: nStations (the number of nodes) and nArcs (the number of arcs).
- We read the .arc file, and build a table containing the information about the arcs. Since our problem does not consider the type of product, we only use the information of the from node, to node, cost and mutual capacity.
  Note that the cost for each arc is set to 1, despite the fact that we do not consider the cost in this problem. Setting it to 1 simply means the number of arcs contained by the path.
- We build a list about the nodes. For each node, the list contains a line indicating the ID, and the outdegree of the node.
- We build a cost table, which contains the cost and the id for for each arc.
- Finally, we read the '.od' file and build a table about the commodities. Each line contains the origin and the destination, and the quantity of a commodity flow.

# 7 Numerical Experiments

## 7.1 Dantzig Wolfe

We only use the AerTrans and JLF Assad datasets. Because only they satisfy our most important needs: all links are bundled. Note that AerTrans datasets may have the flow with unspecified destination. We choose to simply ignore those flows.

We briefly describe the usage of our codes.[1]

1. dijkstra.m: compute the minimal cost path given a initial node.
2. getInitSet.m: finds feasible paths between the OD pair by Dijkstra algorithm,

[1] Codes can be found at https://github.com/brandonchen99/multicommodity-flow/

3. readdata.m: input the data.
4. sup2od.m: for those who do not have an .od file, transform the .sup file into the .od file.
5. restricted_Master.m solves the restricted master problem by "linprog".
6. pricingProblem.m: solves the pricing problem.
7. main.m: main program, column generation iteration.

The numerical results are listed below. We find that for the AerTrans datasets, the most time-cosuming part is finding the initial set, since there are lots of OD pairs. Generally, the time on the column generation (solving the problem) is small.

Table 1: JLF Assad - Danzig Wolfe

| | assad1.5k | assad1.6k | assad3.4k | assad3.7k |
|---|---|---|---|---|
| Loading data | 0.009 | 0.011 | 0.012 | 0.013 |
| Finding initial set | 0.025 | 0.025 | 0.043 | 0.049 |
| Solving problem | 0.083 | 0.098 | 0.208 | 0.198 |
| Total | 0.117 | 0.134 | 0.263 | 0.26 |
| Objective | 1 | 1 | 0.8898 | 0.9153 |

Table 2: AerTrans - Dantzig Wolfe

| | jl023 | jl049 | jl141 | jl147 | jl158 | jl188 | jl207 | jl209 |
|---|---|---|---|---|---|---|---|---|
| Loading data | 0.017 | 0.053 | 0.373 | 0.309 | 0.307 | 0.795 | 1.18 | 0.987 |
| Finding initial set | 0.046 | 0.735 | 5.927 | 5.688 | 6.175 | 11.97 | 38.908 | 37.278 |
| Solving problem | 0.018 | 0.041 | 0.613 | 0.548 | 0.538 | 1.197 | 1.666 | 9.57 |
| Total | 0.081 | 0.829 | 6.913 | 6.545 | 7.02 | 13.962 | 41.754 | 47.835 |
| Objective | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

### 7.2 Lagrange Relaxation

The numerical results are listed below. The time consumed is much more than the Dantzig Wolfe Decomposition. Hnece, we only test it on the JLF Assad datasets, where only a few OD pairs exist.

Table 3: JLF Assad - Lagrange

| | assad1.5k | assad1.6k | assad3.4k | assad3.7k |
|---|---|---|---|---|
| Loading data | 0.007 | 0.021 | 0.013 | 0.005 |
| Total | 12.756 | 14.57 | 16.23 | 20.45 |
| Objective | 1 | 1 | 0.8899 | 0.9153 |

## References


[1] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

[2] Kim L. Jones, Irvin Lustig, Judith M. Farvolden, and Warren B. Powell. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming*, 62:95–117, 1993.