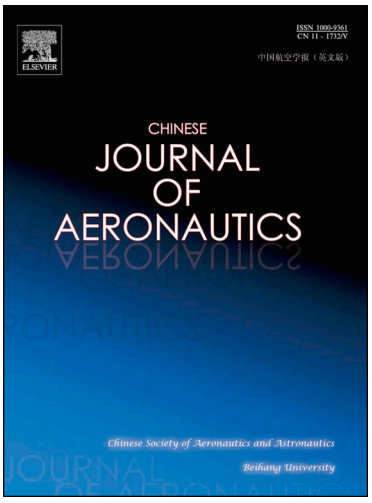
Accepted Manuscript



Full Length Article

On solving multi-commodity flow problems: An experimental evaluation

Weibin Dai, Jun Zhang, Xiaoqian Sun

PII: S1000-9361(17)30122-X
DOI: http://dx.doi.org/10.1016/j.cja.2017.05.012
Reference: CJA 859

To appear in: *Chinese Journal of Aeronautics*

Received Date: 14 June 2016
Revised Date: 8 October 2016
Accepted Date: 21 January 2017

Please cite this article as: W. Dai, J. Zhang, X. Sun, On solving multi-commodity flow problems: An experimental evaluation, *Chinese Journal of Aeronautics* (2017), doi: http://dx.doi.org/10.1016/j.cja.2017.05.012

Chinese Journal of Aeronautics 28 (2015) xx-xx

Contents lists available at ScienceDirect

Chinese Journal of Aeronautics

journal homepage: www.elsevier.com/locate/cja

On solving multi-commodity flow problems: An experimental evaluation

Weibin DAI[a,b,c], Jun ZHANG[a,b,c], Xiaoqian SUN[a,b,c,*]

[a]*School of Electronic and Information Engineering, Beihang University, Beijing 100083, China*
[b]*National Key Laboratory of CNS/ATM, Beijing 100083, China*
[c]*Beijing Key Laboratory for Network-based Cooperative ATM, Beijing 100083, China*

Received 14 June 2016; revised 8 October 2016; 21 January 2017

Abstract

Multi-commodity flow problems can be found in many areas, such as transportation, communication, and logistics. Therefore, such problems have been studied by a multitude of researchers, and a variety of methods have been proposed for solving it. However, most researchers only discuss the properties of different models and algorithms without taking into account the impacts of actual implementation. In fact, the true performance of a method may differ greatly across various implementations. In this paper, several popular optimization solvers for implementations of column generation and Lagrangian relaxation are discussed. In order to test scalability and optimality, three groups of networks with different structures are used as case studies. Results show that column generation outperforms Lagrangian relaxation in most instances, but the latter is better suited to networks with a large number of commodities.

Keywords: Multi-commodity flow problem; column generation; Lagrangian relaxation; evaluation; implementation

1. Introduction[1]

The multi-commodity flow problem (MCFP) deals with the assignment of commodity flows from source to destination in a network. MCFPs are highly relevant in several fields including transportation[1] and telecommunications.[2]

MCFPs have been studied by a number of researchers for several decades, and a variety of solutions have been proposed such as column generation, Lagrangian relaxation, branch-and-bound, and Dantzig-Wolfe decomposition. Tomlin[3] presented a column generation algorithm first, and he is also one of forerunners of using Dantzig-Wolfe decomposition. Based on these methods, a number of new processes were proposed more recently. For instance, Barnhart et al. presented a partitioning solution approach in order to solve large-scale MCFPs with large numbers of commodities.[4] Many constraints can be relaxed with a cycle-based formulation and column generation. Then, by solving a series of linear programs with reduced size, an optimal solution can be obtained within a finite number of steps. Based on column generation, Barnhart et al. presented a modified version of the branch-and-bound algorithm for solving origin-destination integer multi-commodity flow problems.[5] An algorithm for path-based models was

*Corresponding author. *E-mail address:* sunxq@buaa.edu.cn

improved by presenting a new branching rule and adding cuts. Computational complexity can be reduced significantly by these cuts in many real-world situations. The classical column generation algorithm has a variety of advantages for solving MCFPs. However, because the restricted master problem (RMP) changes a lot with continuous column generation iterations, the solution often convergences slowly. Therefore, Gondzio et al. proposed the primal-dual column generation method (PDCGM) for solving this problem.[6-8] PDCGM is a modified method relying on the sub-optimal dual solutions of restricted master problems where solutions are obtained with the primal-dual interior-point method. This method was initially developed for solving large-scale convex optimization problems. PDCGM was shown by the authors to be competitive and suitable for a wide context of optimization problems.

In addition to column generation, new methods based on Dantzig-Wolfe decomposition and Lagrangian relaxation have been developed. Based on Dantzig-Wolfe decomposition, Karakostas proposed polynomial approximation approaches in order to solve MCFPs.[9] These approaches were also derived from other previous algorithms.[10,11] These methods minimize computation time dependence on the number of commodities. Based on Lagrangian relaxation, Retvari et al. proposed a new Lagrangian relaxation method that can solve the MCFP as a sequence of single-commodity flow problems.[12] This technique performs best when solving OSPF (open shortest path first) traffic engineering problems because a given path can be improved towards approximate optimality while giving several necessary parameters. In addition, Babonneau and Vial proposed a new method based on partial Lagrangian relaxation,[13] which constrains relaxation to the set of arcs that are saturated at the optimum. This method can be used to solve large problems.

In addition to the previous research, modified versions of MCFPs are also studied. Moradi et al. proposed a new column generation method for solving bi-objective MCFP problems.[14] Based on the simplex method and Dantzig-Wolfe decomposition, the algorithm moves between different points. Similar to Karakostas's method, it demonstrates that the average computation time does not necessarily depend on the number of commodities. Path-based models are often used to solve MCFPs; However, Bauguion et al. proposed a new idea about such models.[15,16] Instead of generating paths for each commodity, they generated groups of paths in other ways, such as combining them into single sets or separate trees. These methods reduce computation time compared with other models.

MCFPs can be applied to many different application problems.[17] Zhang et al. presented a multi-commodity model for supply chain networks,[18] which was solved using the Benders decomposition method. Caimi et al. studied the problems of conflict-free train routing and scheduling and proposed a new resource-constrained model based on the multi-commodity flow.[19] Morabito et al. studied network routing for generalized queuing networks and presented a multi-commodity flow algorithm based on a routing step and an approximate decomposition step.[20] Shitrit et al. applied the multi-commodity flow problem to tracking of multiple people. Experimental results showed that their approach performs better than other state-of-the-art tracking algorithms.[21]

As our review of related literature shows, the vast majority of the methods proposed for solving MCFPs discussed the properties of different models and algorithms without considering the impact of implementation. For instance, while solving the MCFP with an algorithm, some portions of the problem can be formulated as a linear program. In this case, it is important to select the appropriate implementation type, i.e. linear program solver.[22] This paper introduces the formulations and processes of two commonly-used algorithms for solving MCFPs: column generation and Lagrangian relaxation. In addition to the algorithm theory, implementations for MCFPs are also discussed. Several popular program solvers, such as GNU linear programming kit(GLPK), CVXPY, GUROBI, and SCIPY, are introduced briefly. Past research often performed comparisons with single, outdated competing methods or implementations. This paper promotes the idea of comparing popular methods in order to decide which one is the best for a given problem. In order to test the scalability and optimality of the algorithms and program solvers, three groups of networks (grid, planar, and airport networks) are chosen as case studies. For each group of networks, the optimality, computation time, and number of iterations for the algorithms and different program solvers are compared.

Results indicate that column generation has better properties for solving an MCFP than Lagrangian relaxation in most instances. However, Lagrangian relaxation can be faster and within acceptable optimality bounds in certain large-scale networks with a high number of commodities. There is a similar relationship with implementations for column generation: CVXPY performs better than GLPK for solving MCFPs with a large number of commodities by column generation while GLPK is superior for a small-scale MCFP. This work shows the tremendous impact of implementation techniques on computation time and solution quality and lays the foundation for further research on MCFPs.

The remainder of this paper is organized as follows: An introduction and formulation of the multi-commodity flow problem are provided in Section 2. Relevant solution techniques, such as column generation, Lagrangian relaxation, and several program solvers, are introduced in Section 3. In order to compare these techniques, three groups of different datasets are discussed in Section 4. The evaluations performed using these datasets as case stud-

ies are presented in Section 5. Conclusions are in Section 6.

2. Multi-commodity flow problem

The MCFP seems like a combination of several single-commodity flow problems. However, because of the interaction between commodities, the complexity of MCFP is much higher than that for solving each single-commodity flow problem independently.[23] In order to solve MCFPs, two necessary constraints must be considered. The first is the travel demand, which means that all the commodities need to be transported to their destinations. The second is the edge capacity constraint. This means that the flow on each edge can not exceed flow capacity. The first constraint is essentially the sum of a set of single-commodity flow problems. However, the second constraint needs to consider all commodities together, and it causes interactions between them.

$G=(V,E)$ is a directed network. Here V and E are the set of nodes and edges with sizes n and m, respectively. For each edge l, there is a cost c_l and capacity cap_l. t kinds of commodities need to be transported from their origin nodes to destination nodes. $O(k)$ and $D(k)$ are used to represent origin and destination node of commodity k[24]. In addition, d^k is the travel demand of commodity k. An optimal flow assignment with minimum cost must be found to satisfy the travel demand and capacity constraints for this problem. Thus, the MCFP can be formulated as follows:[25]

$$\text{minimize } z(x)=\boldsymbol{c}^T\sum_{k=1}^{t}\boldsymbol{X}^k \tag{1}$$

$$\text{subject to } \sum_{k=1}^{t}\boldsymbol{X}^k \le \mathbf{cap} \tag{2}$$

$$\boldsymbol{B}\boldsymbol{X}^k=\boldsymbol{b}^k,\ k=1,2,...,t \tag{3}$$

$$\boldsymbol{X}^k \ge \mathbf{0},\ k=1,2,...,t \tag{4}$$

where Eq. (1) is the objective function of total cost. Eqs. (2) and (3) are an edge capacity constraint and node flow equilibrium equation, respectively. Eq. (4) is a non-negative constraint.

Here, $\boldsymbol{X}^k=[x_1,x_2,...,x_m]^{\mathrm{T}}$ with $k=1,2,...,t$ is the flow vector of each edge for commodity k. The vectors of cost and capacity on each edge are represented by $\boldsymbol{c}=[c_1,c_2,...,c_m]^{\mathrm{T}}$ and $\mathbf{cap}=[\text{cap}_1,\text{cap}_2,...,\text{cap}_m]^{\mathrm{T}}$, respectively[26]. The incidence matrix between nodes and edges is indicated by $\boldsymbol{B}=[B_{il}]_{n\times m}$. It has a size of $n\times m$ and can be defined by listing all edges then, for the l th edge (i,j), setting $B_{il}=1$ and $B_{jl}=-1$; $\boldsymbol{b}^k=[b_1^k,b_2^k,...,b_n^k]^{\mathrm{T}}$ with $k=1,2,...,t$ is defined as follows:[27]

$$b_i^k=\begin{cases} d^k & \text{If } i=O(k) \\ -d^k & \text{If } i=D(k) \\ 0 & \text{Otherwise} \end{cases} \tag{5}$$

A small example, shown in Fig.1 (adapted from Ref.[2]), and Table 1 are discussed as a case study. There are 6 nodes, 10 edges and 3 commodities in this network. The number pair on each edge represents the pair of cost and capacity.

For each commodity k, the travel demand must be satisfied. This means that we need to find a set of paths between $O(k)$ and $D(k)$, and their total flows are equal to the travel demand. In addition, due to edge capacity constraints, the complexity of MCFP is much higher than the combination of t single-commodity flow problems.

The optimal solution of this small problem is shown in Table 2, and the optimal value of the objective function is 43. The results in Table 2 can also be represented with the path flows shown in Table 3.

Note that only a few edges and paths contribute to the optimal solution for each commodity. Flows on a majority of edges and paths are zero. For instance, edge (3,5) is not used at all. This property shows the potential for solving MCFPs more efficiently.

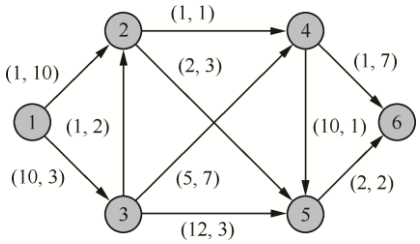


Fig.1 A small example

Table 1 Sample commodities

Commodity k	$O(k)$	$D(k)$	Travel demand d^k
1	1	4	1
2	1	5	3
3	1	6	2

Table 2 Solution I for the small example

Edge	Commodity 1	Commodity 2	Commodity 3
(1,2)	0	3	1
(1,3)	1	0	1
(3,2)	0	0	0
(2,4)	0	0	1
(2,5)	0	3	0
(3,4)	1	0	1
(3,5)	0	0	0
(4,5)	0	0	0
(4,6)	0	0	2
(5,6)	0	0	0

Table 3 Solution II for the small example

Commodity	Path	Flow
1	$1 \to 3 \to 4$	1
2	$1 \to 2 \to 5$	3
3	$1 \to 2 \to 4 \to 6$	1
	$1 \to 3 \to 4 \to 6$	1

3. Solution techniques

Section 2 shows that the formulation of the MCFP has a structure that should be exploited during computation of a solution. Eq. (3) can be divided into t independent equations for each commodity k, directly. Eq. (2) is the only interaction between all the commodities. If the commodities do not influence each other in this constraint, the MCFP can easily be solved through the solutionof several independent single-commodity flow problems. From this concepti, Lagrangian relaxation is proposed for Eq. (2). Just as with the small example in Section 2, most variables do not contributie to the solution. Based on this observation, a column generation algorithm,that only considers a small set of variables is presented. In this section, several techniques including the column generation in Section 3.1, Lagrangian relaxation in Section 3.2, and optimization solvers in Section 3.3 for implementation are introduced.

3.1. Column generation

In Eqs. (1)-(4), there are $t \times m$ variables in the model. However, while solving the multi-commodity flow problem, most variables do not contribute to the optimal solution (see the example in Section 2). In this case, column generation is used to solve MCFP. New columns are added into the solution step by step as necessart to improve the objective function. The process of this algorithm is as follows:

First, a set of initial columns are used to construct the dual problem. The solutions of the dual problem are used to generate price problems for all the commodities. We assume that u_*^k is the optimal objective function value of the k th price problem. If $u_*^k \geq 0$ for $\forall k = 1, 2, ..., t$, the original MCFP reaches the optimal solution.[3] Otherwise,

if $u_*^k < 0$ for $k = k_1, k_2, ..., k_p$, then the solutions of these price problems need to be added as a new column into the set. The algorithm continues running with the steps above until optimal solutions are obtained.

3.1.1. Dantzig-Wolfe decomposition and master problem

A problems is first dealt with using Dantzig-Wolfe decomposition and split into a master problem (containing a set of currently active columns) and a set of sub-problems.

For each commodity k, variable vector $\boldsymbol{X}^k$ can be decomposed as $\boldsymbol{X}^k = \sum_{j=1}^{n_k} \lambda_j^k \boldsymbol{Y}_j^k$. Here, λ_j^k with $j = 1, 2, ..., n_k$ are coefficients satisfying $\sum_{j=1}^{n_k} \lambda_j^k = 1$ and $\boldsymbol{Y}_j^k$ is the set of columns that satisfy the following constraints:

$$\boldsymbol{Y}_j^k \le \mathbf{cap} \tag{6}$$

$$\boldsymbol{B}\boldsymbol{Y}_j^k = \boldsymbol{b}^k \tag{7}$$

$$\boldsymbol{Y}_j^k \ge \mathbf{0} \tag{8}$$

Therefore, the master problem can be formulated as

$$\text{Minimize} \quad z(\lambda) = \sum_{k=1}^{t} \sum_{j=1}^{n_k} (\boldsymbol{c}^{\mathrm{T}} \boldsymbol{Y}_j^k) \lambda_j^k \tag{9}$$

$$\text{subject to} \quad \sum_{k=1}^{t} \sum_{j=1}^{n_k} \boldsymbol{Y}_j^k \lambda_j^k \le \mathbf{cap} \tag{10}$$

$$\sum_{j=1}^{n_k} \lambda_j^k = 1, k = 1, 2, ..., t \tag{11}$$

$$\lambda_j^k \ge 0, \ j = 1, 2, ..., n_k, k = 1, 2, ..., t \tag{12}$$

The optimal solution based on this set of columns can be obtained by solving the master problem. However, at this stage it can not be ensured that this solution is also optimal for the original MCFP. Therefore, the dual problem and price problems are solved in the next section.

3.1.2. Dual problem

In order to show the model clearly, the formulation Eqs. (9)-(12) are transformed into matrix form.

$$\text{Minimize} \quad z(\lambda) = \boldsymbol{c}^{\mathrm{T}} \boldsymbol{Y} \boldsymbol{\lambda} \tag{13}$$

$$\text{subject to} \quad \boldsymbol{Y}\boldsymbol{\lambda} \le \mathbf{cap} \tag{14}$$

$$\boldsymbol{N}\boldsymbol{\lambda} = \mathbf{1}_t \tag{15}$$

$$\boldsymbol{\lambda} \ge \mathbf{0}_{\text{num}} \tag{16}$$

where $\boldsymbol{Y} = [\boldsymbol{Y}_1^1, \boldsymbol{Y}_2^1, ..., \boldsymbol{Y}_{n_1}^1, \boldsymbol{Y}_1^2, \boldsymbol{Y}_2^2, ..., \boldsymbol{Y}_{n_2}^2, ..., \boldsymbol{Y}_1^t, \boldsymbol{Y}_2^t, ..., \boldsymbol{Y}_{n_t}^t]$ and $\boldsymbol{\lambda} = [\lambda_1^1, \lambda_2^1, ..., \lambda_{n_1}^1, \lambda_1^2, \lambda_2^2, ..., \lambda_{n_2}^2, ..., \lambda_1^t, \lambda_2^t, ..., \lambda_{n_t}^t]^{\mathrm{T}}$ are the matrices of selected columns and vectors of coefficients. The number of columns is $\text{num} = \sum_{k=1}^{t} n_k$. $\boldsymbol{N} = (N_{kj})_{t \times \text{num}}$ is the incidence matrix between commodities and columns. $N_{kj} = 1$ if column j is one assignment for commodity k, and $N_{kj} = 0$ otherwise. For instance, if $n_1 = 2$, $n_2 = 3$, and $n_3 = 4$..., then $\boldsymbol{N}$ can be shown as follows:

$$\boldsymbol{N} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & ... & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & ... & 0 \\ & & & & & \vdots & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{17}$$

The vectors of dual variables for Eqs. (14) and (15) are indicated by $\boldsymbol{w} = [w_1, w_2, ..., w_m]^{\mathrm{T}}$ and $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_t]^{\mathrm{T}}$, respectively. Therefore, the dual problem of Eqs. (13)-(16) can be formulated as

$$\text{Minimize} \quad z_{\mathrm{d}}(w, \alpha) = \mathbf{cap}^{\mathrm{T}}\boldsymbol{w} + \mathbf{1}_t^{\mathrm{T}}\boldsymbol{\alpha} \tag{18}$$

$$\text{subject to} \quad \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{N}^{\mathrm{T}}\boldsymbol{\alpha} \le \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{c} \tag{19}$$

$$\boldsymbol{w} \ge \mathbf{0}_m, \boldsymbol{\alpha} \text{ free} \tag{20}$$

3.1.3. Price problems

The optimal value of dual variables $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ can be obtained by solving the dual problem from Eqs. (18-20). In order to make sure of the optimality of the master problem's solution and adding of new columns, the price problems need to be solved. For each commodity k, the price problem is formulated as

$$\text{Minimize} \quad u(\boldsymbol{X}^k) = (\boldsymbol{c} - \boldsymbol{w})^{\mathrm{T}}\boldsymbol{X}^k - \boldsymbol{\alpha}_k \tag{21}$$

$$\text{subject to} \quad \boldsymbol{B}\boldsymbol{X}^k = \boldsymbol{b}^k \tag{22}$$

$$\boldsymbol{X}^k \ge \mathbf{0} \tag{23}$$

We assume that u_*^k is the optimal value of $u(\boldsymbol{X}^k)$ with constraints Eqs. (22)-(23). Then, it is proven[2] that the original MCFP can obtain the optimal solution if $u_*^k \ge 0$ for $\forall k = 1, 2, ..., t$.

If $u_*^k < 0$ for $k = k_1, k_2, ..., k_{\mathrm{p}}$, then $\boldsymbol{X}_*^k$ with $k = k_1, k_2, ..., k_{\mathrm{p}}$ needs to be added into the column set as new columns. A new dual problem and price problems will then be constructed. A loop will continue with these steps until the optimal solution is obtained.

3.2. Lagrangian relaxation

In addition to column generation, Lagrangian relaxation can also be used to solve the MCFP. Lagrange multipliers move towards the optimal value step by step for objective functions closer to optimal. The process is shown as follows:

First, we solve the initial Lagrangian sub-problem with initial Lagrange multipliers. Then, in each iteration, the solution of the sub-problem is used to update the multipliers, and a new solution is obtained by solving the sub-problem with new multipliers. The objective function value will be close enough to the optimal solution if the number of iterations is large enough. The details of this algorithm will be introduced in the following section.

3.2.1. Lagrangian sub-problem

Let $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_m]^{\mathrm{T}}$ denote Lagrange multipliers of Eq. (2), then the Lagrangian sub-problem can be generated as follows:

$$L(\mu) = \text{minimize} \quad \boldsymbol{c}^{\mathrm{T}}\sum_{k=1}^{t}\boldsymbol{X}^k + \boldsymbol{\mu}^{\mathrm{T}}\left(\sum_{k=1}^{t}\boldsymbol{X}^k - \mathbf{cap}\right) \tag{24}$$

$$\text{subject to} \quad \boldsymbol{B}\boldsymbol{X}^k = \boldsymbol{b}^k, k = 1, 2, ..., t \tag{25}$$

$$\boldsymbol{X}^k \ge \mathbf{0}, k = 1, 2, ..., t \tag{26}$$

Equivalently, Eq. (24) can also be formulated as

$$L(\mu) = \text{minimize} \quad (\boldsymbol{c} + \boldsymbol{\mu})^{\mathrm{T}}\sum_{k=1}^{t}\boldsymbol{X}^k - \boldsymbol{\mu}^{\mathrm{T}}\mathbf{cap} \tag{27}$$

Ignoring the fixed term $-\boldsymbol{\mu}^{\mathrm{T}}\mathbf{cap}$, Eq. (27) is reformulated as

$$\sum_{k=1}^{t}(\boldsymbol{c} + \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{X}^k \tag{28}$$

Therefore, the linear program of Eqs. (25), (26) and (27) can be split into t independent sub-problems for each commodity $(k = 1, 2, ..., t)$ as follows:

$$\text{Minimize} \quad \sum_{k=1}^{t} (\boldsymbol{c} + \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{X}^k \tag{29}$$

$$\text{subject to} \quad \boldsymbol{B}\boldsymbol{X}^k = \boldsymbol{b}^k \tag{30}$$

$$\boldsymbol{X}^k \geq \boldsymbol{0} \tag{31}$$

By solving these, the optimal solution for Lagrange multipliers $\boldsymbol{w}$ is obtained. However, this solution may not be optimal for the original MCFP. Therefore, the Lagrangian multiplier problem should be solved.

3.2.2. Lagrangian multiplier problem

It has been proven that the value of Lagrangian function $L(\boldsymbol{w})$ is a lower bound on the optimal objective function value of the original MCFP.[2] Thus, the optimal value z_* for Eqs. (1)-(4) can be formulated as

$$z_* = \max_{\mu \geq 0} \quad L(\boldsymbol{\mu}) \tag{32}$$

$$\text{subject to} \quad \text{Eqs. (24-26)} \tag{33}$$

In order to solve this problem, researchers often use gradient methods to approach the optimal solution. Let $\boldsymbol{X}_* = [\boldsymbol{X}_*^1, \boldsymbol{X}_*^2, ..., \boldsymbol{X}_*^t]$ be the solution for Eqs. (24)-(26). The multipliers $\boldsymbol{w}$ should be updated as

$$\boldsymbol{\mu}^{q+1} = [\boldsymbol{\mu}^q + \theta^q (\sum_{k=1}^{t} \boldsymbol{X}_*^k - \mathbf{cap})]^+ \tag{34}$$

where q is the number of iterations, θ^q is a step size and $[a]^+ = \max(a, 0)$

In the next section, the selection of step size is introduced.

3.2.3. Choosing step size

Suitable step sizes are important for solving the Lagrangian multiplier problem. If they are too small, the algorithm may not converge; if they are too large, the optimal solution may be overshot. Therefore, the following conditions should be satisfied:[2]

$$\theta^q \to 0 \tag{35}$$

$$\sum_{j=1}^{q} \theta^j \to \infty \tag{36}$$

For instance, $\theta^q = \dfrac{1}{q}$ is a simple choice. In this paper, we use the following step size:

$$\theta = \frac{\lambda[\mathrm{UB} - L(\boldsymbol{\mu})]}{\left\| \sum_{k=1}^{t} \boldsymbol{X}_*^k - \mathbf{cap} \right\|} \tag{37}$$

The algorithm of Lagrangian relaxation is shown in Algorithm 1. Its properties will be tested in Section 5.

Algirithm 1: Algorithm of Lagrangian relaxation

Initialize: initial number of iteration $q = 0$, flag=0, $\lambda = 1$, $\boldsymbol{\mu} = \boldsymbol{0}$, initial upper bound and lower bound UB,LB, eps=10^{-5}, max number of iteration is mi.

while $q \leq$ mi and $\lambda >$eps :

solve program Eqs. (29)-(31) and obtain $\boldsymbol{X}_* = [\boldsymbol{X}_*^1, \boldsymbol{X}_*^2, ..., \boldsymbol{X}_*^t], L(\boldsymbol{\mu})$.

if $\boldsymbol{X}_*$ is feasible:

UB=$z(\boldsymbol{X}_*)$

endif

if $L(\boldsymbol{\mu}) <$ LB :

flag = 3

else:

if $L(\boldsymbol{\mu}) - \mathrm{LB} < \mathrm{eps} \times \max(1, \mathrm{LB})$:
flag=flag+1
endif
if $L(\boldsymbol{\mu}) > \mathrm{LB}$:
$\mathrm{LB} = L(\boldsymbol{\mu})$
endif
endif
if flag>2 :
$\lambda = \lambda / 2$
flag=0
endif

$$\theta = \frac{\lambda[\mathrm{UB} - L(\boldsymbol{\mu})]}{\left\| \sum_{k=1}^{t} \boldsymbol{X}_*^k - \mathbf{cap} \right\|}$$

update $\boldsymbol{\mu}$ with Eq. (34), $q = q + 1$
endwhile
Output: The optimal solution for original MCFP is LB

3.3. Implementations

While implementing the column generation algorithm, the master problem in Section 3.1.1, dual problem in Section 3.1.2 and price problems in Section 3.1.3 are all linear programs. They can all be transformed into the following formulation:

$$\text{Minimize} \quad \boldsymbol{c}^{\mathrm{T}} \boldsymbol{X} \tag{38}$$

$$\text{subject to} \quad \boldsymbol{A}\boldsymbol{X} \geq \boldsymbol{b} \tag{39}$$

$$\mathbf{Aeq}\boldsymbol{X} = \mathbf{beq} \tag{40}$$

The Lagrangian sub-problem in Section 3.2.1 for Lagrangian relaxation is also the same. In order to solve this, it is necessary to find an approximate linear program solver.

Recently, a number of optimizers have become more and more popular, such as GLPK, CVXPY, and GUROBI. There is also one linear program solver, SCIPY.optimize.linprog, in Python. They will be introduced briefly in this section.

3.3.1. GLPK

The GLPK is a package for solving large-scale linear (LP), mixed integer (MIP), and related optimization problems.[28]

3.3.2. CVXPY

CVXPY is a modeling language embedded in Python for solving convex optimization problems. With this language, the problem can be expressed in a natural way similar to math formulations instead of the standard forms for solvers.[29]

3.3.3. GUROBI

GUROBI is a commercial optimization solver for linear, quadratic, mixed integer, and several other related programs. Similar to GLPK and CVXPY, it can support a series of program and modeling languages including C++, Java, and Python.[30]

3.3.4. SCIPY

SCIPY.optimize.linprog is a built-in linear program solver in Python. By inputting the values of objective vectors and constraint matrices in the standard form, linear programs can be solved with this tool [31].

Note that Lagrangian sub-problem from Eqs. (29)-(31) for each commodity k is in fact a shortest-path problem. The only difference is that the cost vector $\boldsymbol{c}$ is add by Lagrangian multiplier μ. Thus, the Dijkstra method for finding the shortest path from $O(k)$ to $D(k)$ can be used in addition to the presented program solvers. The speeds of the Dijkstra method and program solvers all depend on the network size, so their properties will be compared with different kinds of networks in Section 5.

4. Datasets

In order to test the scalability and quality of algorithms and program solvers, three groups of networks, grid networks, planar networks, and airport networks, are chosen as case studies.

The structure of these networks are shown in Tables 4-6. The parameters in the first line are explained as follows: n, m, t are the numbers of nodes, edges, and commodities; p is the edge percentage of all possible edges, i.e. $p = \dfrac{2m}{n(n-1)}$; ADG, ASPLC, ASPLA are average degree, average shortest path length of all the commodities, and average shortest path length of all the node pairs, respectively.

4.1. Grid networks

Table 4 Grid networks

Instance	n	m	p(%)	t	ADG	ASPLC	ASPLA
Grid 1	25	80	26.67	50	6.4	3.2	3.2
Grid 2	25	80	26.67	100	6.4	3.3	3.2
Grid 3	100	360	7.27	50	7.2	6.4	6.6
Grid 4	100	360	7.27	100	7.2	6.5	6.6
Grid 5	225	840	3.33	100	7.5	10.2	10.0
Grid 6	225	840	3.33	200	7.5	10.4	10.0
Grid 7	400	1520	1.90	400	7.6	13.5	13.3
Grid 8	625	2400	1.23	500	7.7	17.2	16.6
Grid 9	625	2400	1.23	1000	7.7	16.8	16.6

Nine grid networks are shown in Table 4 with n ranging from 25 to 625. With increase of network size, the average degrees all remain small values. Thus, the ASPLC and ASPLA both become larger and larger. this means that for a larger grid network, longer paths are needed to connect two arbitrary nodes on average.

4.2. Planar networks

The properties of five planar networks are shown in Table 5. Average degree values always remain large compared with grid networks. Thus, short paths can be used to connect two nodes. In addition, the commodity numbers t of planar networks are much larger than those in grid networks. This means that more commodities with different OD (origin-destination) pairs need to be transported.

Table 5 Planar networks

Instance	n	m	p(%)	t	ADG	ASPLC	ASPLA
Planar 1	30	150	34.48	92	10.0	2.7	2.5
Planar 2	50	250	20.41	267	10.0	3.6	3.5
Planar 3	80	440	13.92	543	11.0	4.0	3.9
Planar 4	100	532	10.75	1085	10.6	4.5	4.6
Planar 5	150	850	7.61	2239	11.3	5.2	5.1

4.3. Airport networks

In addition to the two groups of networks above, several airport networks from different countries are also discussed. As shown in Table 6, the degree average is high, and most OD pairs can be connected in paths with two edges (ASPLC < 2). This means that nodes in these networks are strongly connected, which is quite different from grid and planar networks.

Table 6 Airport networks

Instance	n	m	p(%)	t	ADG	ASPLC	ASPLA
Germany	26	162	49.85	191	12.5	1.2	1.9
France	46	314	30.34	499	13.7	1.4	2.1
Spain	42	361	41.93	586	17.2	1.4	1.9
Australia	124	455	5.97	1240	7.3	1.7	3.0
India	74	437	16.18	1553	11.8	1.7	2.3
China	183	2995	17.98	4552	32.7	1.3	2.2

5. Evaluation

In this section, the properties of column generation and Lagrangian relaxation with different program solvers are tested on several kinds of networks. In order to compare the quality of each algorithm, a parameter gap is defined as follows:

$$\text{gap} = \frac{|z_* - z_{\text{opt}}|}{z_{\text{opt}}} \tag{41}$$

where z_* is the objective function value of the solution with each algorithm, and z_{opt} is the optimal value of the objective function. In addition to the quality, the computation time and number of iterations are also discussed in this section.

5.1. Grid networks

The gaps of grid networks are shown in Table 7 and Table 8. Overall, the quality of column generation is better than that of Lagrangian relaxation. As shown in Table 7, the solutions obtained by column generation are all optimal except for Grid 5 with SCIPY. However, in Table 8, the solutions obtained by the five implementations for Lagrangian relaxation are within 1% of optimality incases where that they can obtain solutions.

The computation times for column generation and Lagrangian relaxation are shown in Fig.2. The speed of column generation is faster than that of Lagrangian relaxation with each implementation. For column generation, the run time ranking of four implementations is GLPK<GUROBI<CVXPY<<SCIPY . For Lagrangian relaxation, it is Dijkstra<GLPK<CVXPY<GUROBI<<SCIPY . In both algorithms, SCIPY has the worst results, and column generation with GLPK is the best choice for solving grid problems.

The number of iterations are reported in Table 9 and Table 10. It is shown that, for each network, the iteration numbers of column generation are much less than those of Lagrangian relaxation. This may be the major reason for the difference between their computation times.

Table 7 Gaps of grid problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
Grid 1	0	0	0	0
Grid 2	0	0	0	0
Grid 3	0	0	0	0
Grid 4	0	0	0	0
Grid 5	0	0	0	0.056%
Grid 6	0	0	0	*
Grid 7	0	0	0	*
Grid 8	0	0	0	*
Grid 9	0	*	0	*

Note: * represents cases where the run time is over 300 s.

Table 8 Gaps of grid problems with Lagrangian relaxation

Instance	Dijkstra (%)	GLPK (%)	CVXPY (%)	GUROBI (%)	SCIPY (%)
Grid 1	0.013	0.013	0.013	0.016	0.013
Grid 2	0.932	0.357	0.360	1.124	0.382
Grid 3	0.012	0.012	0.007	0.016	0.009
Grid 4	0.010	0.016	0.023	0.010	0.007
Grid 5	0.019	0.011	0.013	0.008	*
Grid 6	0.074	0.047	0.080	0.100	*
Grid 7	0.013	0.011	0.009	0.009	*
Grid 8	0.110	0.024	0.038	*	*
Grid 9	0.022	*	*	*	*

Note: * represents cases where the run time is over 300 s.

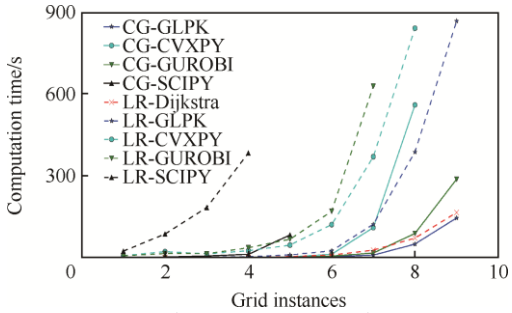


Fig.2 Comparison of computation time for grid networks.

Table 9 Number of grid problem iterations with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
Grid 1	4	4	4	4
Grid 2	4	5	4	4
Grid 3	4	5	4	4
Grid 4	5	6	5	5
Grid 5	7	10	7	7
Grid 6	12	12	12	*
Grid 7	11	13	11	*
Grid 8	17	18	16	*
Grid 9	15	*	16	*

Note: * represents cases where the run time is over 300 s.

Table 10 Number of grid problem iterations with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
Grid 1	44	44	44	54	44
Grid 2	78	73	70	76	79
Grid 3	50	50	55	48	50
Grid 4	53	53	48	54	50
Grid 5	53	52	52	55	*
Grid 6	72	64	68	66	*
Grid 7	62	64	64	68	*
Grid 8	76	76	74	*	*
Grid 9	87	84	*	*	*

Note: * represents cases where the run time is over 300 s.

5.2. Planar networks

In Table 11 and Table 12, the gaps of planar networks are shown. Similar to the results for grid networks, solutions obtained by column generation are all optimal except from Planar 3 with SCIPY. Note that the gap of Planar 3 with SCIPY in Table 11 is 51%, and the solution in this case is wrong. This shows that SCIPY is not reliable while solving large-scale linear programs because of its simplicity. The gaps with Lagrangian relaxation in Table 12 are all within 1% of optimality, but are worse than column generation.

The computation time for planar networks is reported in Fig.3. The run time of column generation is shorter than that of Lagrangian relaxation. Note that the speed of CVXPY is much faster than GLPK with column generation for Planar5 while their run times are similar for Planar1-4. It is shown that column generation with CVXPY is the best for solving large-scale networks with similar structure to planar networks.

The number of iterations are shown in Table 13 and Table 14. Column generation needs many less iterations than Lagrangian relaxation, which is similar to the results of grid networks.

Table 11. Gaps of planar problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY(%)
Planar 1	0	0	0	0
Planar 2	0	0	0	0
Planar 3	0	0	0	51
Planar 4	0	0	0	*
Planar 5	0	0	0	*

Note: * represents cases where the run time is over 800 s.

Table 12. Gaps of planar problems with Lagrangian relaxation

Instance	Dijkstra (%)	GLPK (%)	CVXPY (%)	GUROBI (%)	SCIPY (%)

Planar 1	0.276	0.276	0.276	0.276	0.276
Planar 2	0.324	0.324	0.324	0.324	0.324
Planar 3	0.288	0.288	0.288	0.288	*
Planar 4	0.213	0.166	0.165	0.213	*
Planar 5	0.903	0.903	*	*	*

Note: * represents cases where the run time is over 800 s.

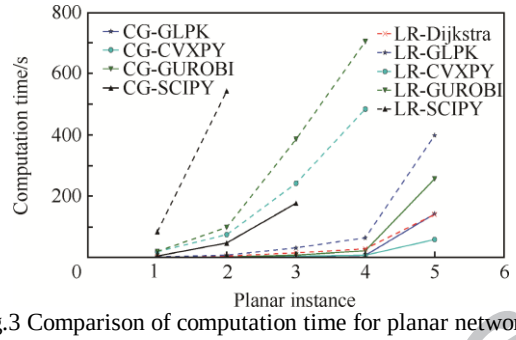


Fig.3 Comparison of computation time for planar networks.

Table 13 Number of planar problem iterations with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
Planar 1	4	4	5	5
Planar 2	6	6	6	6
Planar 3	8	8	8	7
Planar 4	8	8	9	*
Planar 5	10	10	11	*

Note: * represents cases where the run time is over 800 s.

Table 14 Numbers of planar problem iterations with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
Planar 1	52	52	52	52	52
Planar 2	61	61	61	61	61
Planar 3	72	72	72	72	*
Planar 4	60	60	65	60	*
Planar 5	93	93	*	*	*

Note: * represents cases where the run time is over 800 s.

5.3. Airport networks

As shown in Table 15 and Table 16, the gaps of airport networks with column generation are generally smaller than those with Lagrangian relaxation. The quality of column generation is still better than Lagrangian relaxation.

Computation times are shown in Fig.4. The labels DE, FR, ES, AU, IN on *x*-axes represent the five countries Germany, France, Spain, Australia, and India. Time for solving the MCFP for the airport network of China, which is much larger than other networks, is shown in Table 17. Label * indicates that the run time is much longer than other implementations. Note that implementations of Lagrangian relaxation are slower than column generation for first five countries, but, in Table 17, the speeds of Lagrangian relaxation with Dijkstra and GLPK exceed the speed of column generation with most of the implementations. Their performances are also close to column generation with CVXPY.

For column generation, with the increase of network size and commodity number, the speed of CVXPY outperforms GLPK, which is the fastest implementation. CVXPY times are much shorter than GLPK for the airport network of China, as shown in Table 17. Taken together with the results of planar networks, CVXPY can have good properties while solving MCFPs with a large number of commodities.

The number of iterations are shown in Table 18 and Table 19, and the result is similar to the other two groups of networks whose column generation needs less iterations.

Table 15 Gaps of airport networks with column generation

Country	GLPK	CVXPY(%)	GUROBI	SCIPY(%)
Germany	0	0	0	0
France	0	0	0	0
Spain	0	0	0	0.100
Australia	0	0	0	*
India	0	0	0	*
China	0	0.00004	0	*

Note: * represents cases where the run time is over 1800 s.

Table 16 Gaps of airport networks with Lagrangian relaxation

Country	Dijkstra (%)	GLPK (%)	CVXPY (%)	GUROBI (%)	SCIPY (%)
Germany	0.117	0.117	0.117	0.117	0.117
France	0.267	0.267	0.267	0.267	0.267
Spain	0.196	0.196	0.196	0.196	0.196
India	0.349	0.349	0.338	0.349	*
Australia	0.547	0.547	0.547	0.547	*
China	0.068	0.068	*	*	*

Note: * represents cases where the run time is over 1800 s.

Table 17 Computation time for the airport network of China

Algorithm	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
Column generation	-	5186.339	4481.724371	5967.346284	*
Lagrangian relaxation	4484.097	4821.691	*	*	*

Note: * represents cases where the run time is over 1800 s.

Table 18. Number of airport network iterations with column generation

Country	GLPK	CVXPY	GUROBI	SCIPY
Germany	6	103	5	6
France	8	6	13	8
Spain	8	20	10	5
Australia	6	7	9	*
India	9	15	7	*
China	21	13	10	*

Note: * represents cases where the run time is over 1800 s.

Table 19. Numbers of airport network iterations with Lagrangian relaxation

Country	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
Germany	18	18	18	18	18
France	26	26	26	26	26
Spain	56	56	56	56	56
Australia	21	21	21	21	*
India	84	84	91	84	*
China	18	18	*	*	*

Note: * represents cases where the run time is over 1800 s.

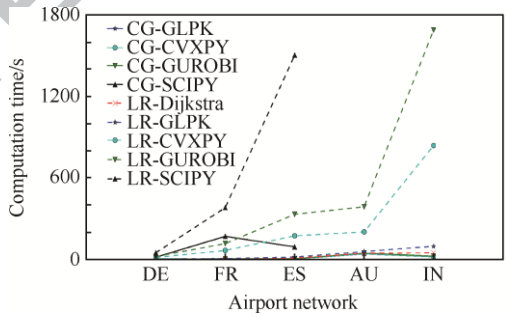


Fig.4 Comparison of computation times for airport networks.

5.4. Summary

Column generation has better properties than Lagrangian relaxation for solving MCFPs in general. However, in large networks with a high number of commodities (like the airport network of China), the best performances of Lagrangian relaxation are close with the column generation. A summary of these implementations for the two algorithms is shown in Table 20 and Table 21.

Table 20 Summary of implementations for column generation

Implementation	Gap	Execution speed	Numbers of iterations
GLPK	0 if solution is obtained	Fast	Small
CVXPY	0 if solution is obtained	Very fast for large commodity Numbers	Small but occasionally large
GUROBI	0 if solution is obtained	Medium speed	Small
SCIPY	Large if network size is large	Slowest	Small

Table 21 Summary of implementations for Lagrangian relaxation

Implementation	Gaps	Execution speed	Numbers of iterations
Dijkstra	Less than 1% if solution is obtained	First	Large
GLPK	Less than 1% if solution is obtained	Second	Large
CVXPY	Less than 1% if solution is obtained	Third	Large
GUROBI	Less than 1% if solution is obtained	Fourth	Large
SCIPY	Less than 1% if solution is obtained	Last	Large

6. Conclusions

In this paper, several algorithms (column generation, Lagrangian relaxation) and implementations (Dijkstra, GLPK, CVXPY, GUROBI, SCIPY) are discussed for solving multi-commodity flow problems. In order to evaluate the scalability and quality of the algorithms with several implementations, three groups of networks, grid, planar and airport, are selected as case studies.

During evaluation, the objective function gaps, computation time, and numbers of iterations for different implementations are compared. It is shown that, in general, column generation performs better than Lagrangian relaxation with all the three evaluation parameters for solving MCFPs. However, for large networks with high number of commodities (like the airport network of China), Lagrangian relaxation is faster than column generation.

Separately, for column generation, GLPK has the best properties, but CVXPY can outperform GLPK while solving MCFPs with a large number of commodities. For Lagrangian relaxation, it is shown that using the Dijkstra shortest-path method to solve the Lagrangian sub-problem is the best choice. In general, GUROBI performs at a medium level in both algorithms, and SCIPY is always the worst.

This paper lays out a baseline for evaluating implementations of MCFP algorithms. For this preliminary evaluation, two algorithms and several implementations are compared. However, several other commonly-used algorithms (such as branch-and-bound) and optimization solvers are not evaluated. For future work, these solution techniques need to be considered as well, and the size of datasets should also be increased further based on the results of our study. In addition, the outcomes of this study can also be applied to certain other problems, such as different types of hub location problems. Because the performance of implementations could vary with the different scales of networks in this paper, the results for solving other problems is expected to be similar.

Acknowledgements

This study was supported by research funds from the National Natural Science Foundation of China (Grant Nos. 61521091, 61650110516, 61601013).

References

1. Sun XQ, Wandelt S, Linke F. Temporal evolution analysis of the european air transportation system: Air navigation route network and airport network. *Transportmetrica B: Transport Dynamics* 2015; 3(2):153–68.
2. Ahuja RK, Magnanti TL, Orlin JB. *Network flows: Theory, algorithms, and applications*. Upper Saddle River: Prentice Hall; 1993. p. 649-94.
3. Tomlin J. Minimum-cost multicommodity network flows. *Operations Research* 1966; 14(1):45–51.
4. Barnhart C, Hane CA, Johnson EL, Sigismondi G. A column generation and partitioning approach for multi-commodity flow problems. *Telecommunication Systems* 1994; 3(3):239–58.
5. Barnhart C, Hane CA, Vance PH. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research* 2000; 48(2):318–26.
6. Gondzio J, Gonzalez-Brevis P, Munari P. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation* 2016, 8(1):47-82.
7. Gondzio J, Gonzalez-Brevis P, Munari P. New developments in the primalcdual column generation technique. *European Journal of Operational Research* 2013; 224(1):41–51.
8. Gondzio J, Gonzalez-Brevis P. A new warmstarting strategy for the primal-dual column generation method. *Mathematical Programming* 2014; 152(1):1–34.
9. Karakostas G. Faster approximation schemes for fractional multicommodity flow problems. *ACM Transactions on Algorithms* 2008; 4(1):166-73.
10. Garg N, Konemann J. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *39th Annual symposium on foundations of computer science;* 1998.p. 300.

11. Fleischer LK. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics* 2000; 13(4):505–20.
12. Retvari G, Biro JJ, Cinkler T. A novel lagrangian-relaxation to the minimum cost multicommodity flow problem and its application to OSPF traffic engineering. *International symposium on computers and communications*, 2004. p. 957–62.
13. Babonneau F, Vial JP. Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. *Operations Research* 2006; 54(1):184–97.
14. Moradi S, Raith A, Ehrgott M. A bi-objective column generation algorithm for the multi-commodity minimum cost flow problem. *European Journal of Operational Research* 2015; 244(2):369–78.
15. Bauguion PO, Ben-Ameur W, Gourdin E. A new model for multicommodity flow problems, and a strongly polynomial algorithm for single-source maximum concurrent flow. *Electronic Notes in Discrete Mathematics* 2013; 41:311–318.
16. Bauguion PO, Ben-Ameur W, Gourdin E. Efficient algorithms for the maximum concurrent flow problem. *Networks* 2015; 65(1):56–67.
17. Cortes P, Munuzuri J, Guadix J, Onieva L. Optimal algorithm for the demand routing problem in multicommodity flow distribution networks with diversification constraints and concave costs. *International Journal of Production Economics* 2013;146(1):313–24.
18. Zhang BW, Qi LQ, Yao EY. A multi-commodity production and distribution model in supply chain. *2010 8th international conference on supply chain management and information systems (SCMIS)*;2010 Oct. 6-9; Hong Kong. Piscataway(NJ):IEEE Press; 2011. p. 1–5.
19. Caimi G, Chudak F, Fuchsberger M, Laumanns M, Zenklusen R. A new resource-constrained multicommodity flow model for conflict-free train routing and scheduling. *Transportation Science* 2011;45(2):212–27.
20. Morabito R, de Souza MC, Vazquez M. Approximate decomposition methods for the analysis of multicommodity flow routing in generalized queuing networks. *European Journal of Operational Research* 2014;232(3):618–29.
21. Shitrit HB, Berclaz J, Fleuret F, Fua P. Multi-commodity network flow for tracking multiple people. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2014;36(8):1614–27.
22. Wandelt S, Sun XQ. Efficient compression of 4d-trajectory data in air traffic management. *IEEE Transactions on Intelligent Transportation Systems* 2015;16(2):844–53.
23. Barnhart C, Krishnan N, Vance PH. *Multicommodity flow problems.* New York: Springer US; 2001 p. 292-302.
24. Meng LH, Xu XH, Zhao YF. Cooperative coalition for formation flight scheduling based on incomplete information. *Chinese Journal of Aeronautics* 2015; 28(6):1747-57.
25. Jordi C, Jordi C. Improving an interior-point algorithm for multicommodity flows by quadratic regularizations. *Networks* 2012;59(1):117–31.
26. Ye BJ, Sherry L, Chen CH, Tian Y. Comparison of alternative route selection strategies based on simulation optimization. *Chinese Journal of Aeronautics* 2016; 29(6):1749-61.
27. Gendron B, Larose M. Branch-and-price-and-cut for large-scale multicommodity capacitated fixed-charge network design. *Euro Journal on Computational Optimization* 2014;2(1-2):55–75.
28. Gnu.org [Internet]. Boston: GLPK - GNU Projection – Free Software Fundation, Inc.; [updated 2012 Jun 23; cited 2017 Apr 28]. Available from: http://www.gnu.org/software/glpk/.
29. Diamond S, Chu E, Boyd S [Internet]. Berlin: CVXPY 0.4.0 documentation, Inc.; [updated 2014 Jan 27; cited 2017 April 28]. Available from http://www.cvxpy.org/en/latest/.
30. Gurobi Optimization [Internet]. Houston: Gurobi Optimization – The Best Mathematical Programming Solver, Inc.; c2015-2017 [updated 2017 Mar 22; cited 2017 Apr 28]. Available from http://www.gurobi.com.
31. Scipy.org [Internet]. [updated 2017 Mar 18; cited 2017 April 28]. Available from http://www.scipy.org.