Final Accepted Version 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, 100191 Beijing, China*
[b]*National Key Laboratory of CNS/ATM, 100191 Beijing, China*
[c]*Beijing Key Laboratory for Network-based Cooperative ATM, Beijing 100191, China*

Abstract

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

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

1. Introduction

The multi-commodity flow problem (MCFP) deals with the assignment of commodity flows from sources to destinations in a network. MCFP is highly relevant in several fields, such as transportation [1] and telecommunication [2].

MCFP has been studied by a number of researchers for several decades and a variety of methods have been proposed, such as column generation, Lagrangian relaxation, branch-and-bound, and Dantzig-Wolfe decomposition. Tomlin presented column generation algorithm first [3], and he is also one of forerunners for using Dantzig-Wolfe decomposition. Based on these methods, a number of new methods have been proposed more recently. For instance, Barnhart et al. presented a partitioning solution approach in order to solve large-scale MCFP with large number of commodities [4]. A large number of constraints can be relaxed with a cycle-based formulation and column generation. Then, by solving a series of linear programs with reduced size, the optimal solution can be obtained within a finite number of steps. Based on column generation, Barnhart et al. presented a modified version of branch-and-bound algorithm for solving origin-destination integer multi-commodity flow problems [5]. An algorithm for path-based model was improved by presenting a new branching rule and adding cuts. The computational complexity can be reduced significantly by the cuts in many real-world situations. The classical column generation algorithm has a variety of advantages for solving MCFP. However, because the restricted master problem (RMP) changes a lot with the continuous column generation iteration, the solution often convergences slowly. Therefore, Gondzio et al. proposed the

*Corresponding author. Tel.: +86 10 82338036
E-mail address: daiweibin@buaa.edu.cn (Weibin Dai),
buaazhangjun@vip.sina.com (Jun Zhang),
sunxq@buaa.edu.cn (Xiaoqian Sun)

primal-dual column generation method (PDCGM) for solving this problem [6,7,8]. PDCGM is a modified method relying on the sub-optimal dual solutions of restricted master problems where the solutions are obtained with the primal-dual interior-point method. This method was developed for solving large-scale convex optimization problems initially. It was shown by the authors that PDCGM is competitive and suitable for a wide context of optimization problems.

In addition to column generation, some new methods based on Dantzig-Wolfe decomposition and Lagrangian relaxation were developed. Based on Dantzig-Wolfe decomposition, Karakostas proposed polynomial approximation approaches in order to solve MCFP [9]. The approaches were also based on some previous algorithms [10,11]. With these methods, the computation time depended least on the number of commodities. Based on Lagrangian relaxation, Retvari et al. proposed a new Lagrangian relaxation method, which can solve MCFP as a sequence of single-commodity flow problems [12]. The technique performs best for solving OSPF (Open Shortest Path First) traffic engineering problems, because if a path is given, it can be improved towards approximate optimality while giving several necessary parameters. In addition, Babonneau et al. proposed a new method based on the partial Lagrangian relaxation [13] and the relaxation is constrained to the set of arcs which are saturated at the optimum. This method can be used to solve large problems.

Besides previous research, some modified versions of MCFP 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 was shown in the evaluation that the average computation time does not depend on the number of commodities necessarily. In order to solve MCFP, path-based models were often used. However, Pierre-Olivier et al. proposed a new idea about the path-based model [15,16]. Instead of generating paths for each commodity, they generated groups of paths in suitable ways. For instance, the paths can be aggregated into trees, into a single set, or into several trees. Compared with other models, it was shown that the computation time can be reduced with their methods.

In addition, MCFP can be applied to many different application problems [17]. Zhang et al. presented a multi-commodity model for supply chain network [18]. The Benders decomposition method was used to solve the problem. 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. Based on a routing step and an approximate decomposition step, they presented a multi-commodity flow algorithm [20]. Shitrit et al. applied multi-commodity flow problem to tracking multiple people. The experimental results showed that their approach performs better than other state-of-the-art tracking algorithms [21].

As our summary of related works shows, a variety of methods have been proposed for solving MCFP. However, the researchers mostly discussed the properties of different models and algorithms without considering the contributions of an implementation. For instance, while solving MCFP with an algorithm, some portions of the problem can be formulated as a linear program. In this case, it is important to select an appropriate implementation, i.e. linear program solver [22]. In this paper, the formulations and processes of two commonly-used algorithms, column generation and Lagrangian relaxation, for solving MCFP, are introduced. In addition to the algorithm theory, implementations for MCFP are also discussed. Several popular program solvers, such as GLPK, CVXPY, GUROBI, and SCIPY, are introduced briefly. In order to test the scalability and optimality of the algorithms and program solvers, three groups of networks (grid networks, planar networks, and airport networks) are chosen as case studies. In different groups of networks, the optimality, computation time, and numbers of iterations for the algorithms with different program solvers are compared.

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

The remainder of this paper is organized as follows. The introduction and formulation of 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. Then, the evaluation with these datasets as case studies is presented in Section 5. The paper concludes with Section 6.

2. Multi-commodity flow problem

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 MCFP, two necessary constraints must be considered. The first one is the travel demand constraint. It means that all the commodities need to be transported to their destinations. The other one is the edge capacity constraint. It means that the flows on each edge can not exceed the 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 the commodities together and it causes the interaction between commodities.

Table 1: Table of notations

Notation	Definition
n, m, t	number of nodes, edges, commodities
n_k	number of columns for commodity k
num	number of all the columns ($num = \sum_{k=1}^{t} n_k$)
c	edge cost vector in which c_l is cost of edge l for $l = 1, 2, ..., m$
cap	edge capacity vector in which cap_l is capacity of edge l for $l = 1, 2, ..., m$
B	incidence matrix between nodes and edges
b^k	node flow equilibrium parameter for commodity k
X^k	flow vector for commodity k in which x_l^k is flow on edge l for $l = 1, 2, ..., m$
Y_j^k	the j th column for commodity k
λ_j^k	coefficients for Y_j^k satisfying $\sum_{j=1}^{n_k} \lambda_j^k = 1$
N	incidence matrix between commodities and columns
w, α	vectors of dual variables for master problem of column generation
μ	vector of Lagrange multipliers
LB	lower bound of objective function in Lagrangian relaxation
θ	step size of μ in each iteration

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

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

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

$$BX^k = b^k, k = 1, 2, ..., t \tag{3}$$

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

where Equation (1) is the objective function of total cost. Equations (2) and (3) are edge capacity constraint and node flow equilibrium equation, separately. Equation (4) is non-negative constraint.

Here, $X^k = (x_1^k, x_2^k, ..., x_m^k)^T$ 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 $c = (c_1, c_2, ..., c_m)^T$ and $cap = (cap_1, cap_2, ..., cap_m)^T$, respectively. The incidence matrix between nodes and edges is indicated by $B = (B_{il})_{n*m}$. It has a size of $n * m$ and can be defined as follows. We put all edges into a list. Then for the l-th edge, we set $B_{il} = 1$ and $B_{jl} = -1$. Moreover, $b^k = (b_1^k, b_2^k, ..., b_n^k)^T$ with $k = 1, 2, ..., t$ is defined as follows [25]:

$$b_i^k = \begin{cases} d^k, if\ i = O(k) \\ -d^k, if\ i = D(k) \\ 0, otherwise \end{cases} \tag{5}$$

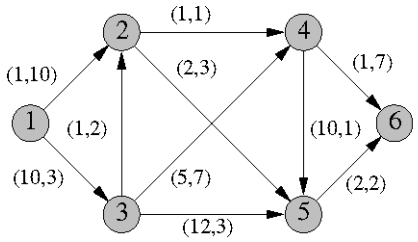

Figure 1: A small example

Table 2: Commodities of the example

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

Table 3: Solution I of a small example

Edges	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 4: Solution II of a small example

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

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

For each commodity k, the travel demand must be satisfied. It 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 the 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 3 and the optimal value of the objective function is 43. The results in Table 3 can also be represented with path flows as shown in Table 4.

Note that only 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 MCFP more efficiently.

3. Solution techniques

In Section 2, it is shown that the formulation of MCFP has a structure that should be exploited during the computation of a solution. Equation (3) can be divided into t independent equations for each commodity k, directly. Equation (2) is the only interaction between all the commodities. If the commodities do not influence each other in this constraint, MCFP can be solved by solving several independent single-commodity flow problems easily. With this conception, Lagrangian relaxation is proposed to relax Equation (2). Besides, just like the small example in Section 2, most variables do not have contributions to the solution. Based on this observation, the column generation algorithm, that only considers a small set of variables, is presented. In this section, several solution techniques including column generation in Section 3.1, Lagrangian relaxation in Section 3.2, and some optimization solvers in

Section 3.3 for implementations are introduced.

3.1. Column generation

In Equations (1--4), it is shown that there are $t*m$ 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 necessary columns are added into solution step by step in order to improve the objective function. The process of this algorithm is shown 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 into the column set as new column. The algorithm runs with the steps above, until the optimal solutions are obtained.

3.1.1. Dantzig-Wolfe decomposition and master problem

First, the problem is dealt with 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 X^k can be decomposed as $X^k = \sum_{j=1}^{n_k} \lambda_j^k Y_j^k$. Here, λ_j^k with $j = 1, 2, ..., n_k$ are coefficients satisfying $\sum_{j=1}^{n_k} \lambda_j^k = 1$ and Y_j^k is called columns that satisfy the following constraints:

$$Y_j^k \leq cap \tag{6}$$

$$BY_j^k = b^k \tag{7}$$

$$Y_j^k \geq 0 \tag{8}$$

Therefore, the master problem can be formulated as follows:

$$\text{Minimuize} \quad z(\lambda) = \sum_{k=1}^{t} \sum_{j=1}^{n_k} (c^T Y_j^k) \lambda_j^k \tag{9}$$

$$\text{subject to} \quad \sum_{k=1}^{t} \sum_{j=1}^{n_k} Y_j^k \lambda_j^k \leq cap \tag{10}$$

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

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

The optimal solution based on this set of columns can be obtained by solving the master problem. However, until 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 (9--12) are transformed into matrix forms.

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

$$\text{subject to} \quad Y\lambda \leq cap \tag{14}$$

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

$$\lambda \geq 0_{num} \tag{16}$$

where $Y = (Y_1^1, ..., Y_{n_1}^1, ..., Y_1^t, ..., Y_{n_t}^t)$ and $\lambda = (\lambda_1^1, ..., \lambda_{n_1}^1, ..., \lambda_1^t, ..., \lambda_{n_t}^t)^T$ are the matrix of selected columns and

vector of coefficients. The number of column is $num = \sum_{k=1}^{t} n_k$. $N = (N_{kj})_{t*num}$ 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 N can be shown as follows:

$$N = \begin{bmatrix} 1100000000...0 \\ 0011100000...0 \\ 0000011110...0 \\ ... \\ 0000000000...1 \end{bmatrix} \tag{17}$$

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

$$\text{Maximize} \quad z_d(w, \alpha) = cap^T w + 1_t^T \alpha \tag{18}$$

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

$$w \ge 0_m, \alpha \ \textit{free} \tag{20}$$

3.1.3. Price problems

The optimal value of dual variables w and α can be obtained by solving the dual problem (18--20). Then, in order to make sure the optimality of the master problem's solution and for adding new columns, the price problems need to be solved. For each commodity k, the price problem is formulated as follows:

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

$$\text{subject to} \quad BX^k = b^k \tag{22}$$

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

We assume that u_*^k is the optimal value of $u(X^k)$ with constraints (22--23). Then, it has been proved [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 $k = k_1, k_2, ..., k_p$, then X_*^k with $k = k_1, k_2, ..., k_p$ need to be added into the column set as new columns. New dual problem and price problems will be constructed based on them. With these steps, the loop will continue until obtaining the optimal solution.

3.2. Lagrangian relaxation

In addition to column generation, Lagrangian relaxation can also be used to solve MCFP. Lagrange multipliers move towards the optimal value step by step for objective functions closer to the optimal one. 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 new solution is obtained by solving the sub-problem with new multipliers. At last, the objective function value is close enough to the optimal solution if the number of iteration is large enough. The details of this algorithm will be introduced in this section.

3.2.1. Lagrangian sub-problem

Let $\mu = (\mu_1, \mu_2, ..., \mu_m)^T$ denote Lagrange multipliers of Equation (2), then Lagrangian sub-problem can be generated as follows:

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

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

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

Equivalently, Equation (24) can also be formulated as follows:

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

Ignoring the fixed term $-\mu^T cap$, Equation (27) is reformulated as:

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

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

$$\text{Minimize} \quad (c+\mu)^T X^k \tag{29}$$

$$\text{subject to} \quad BX^k = b^k \tag{30}$$

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

By solving them, the optimal solution for Lagrange multipliers w is obtained. However, this solution may be not optimal for the original MCFP. Therefore, the Lagrangian multiplier problem should be solved.

3.2.2. Lagrangian multiplier problem

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

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

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

In order to solve this problem, researchers often use gradient methods to approach the optimal solution. Let $X_* = (X_*^1, X_*^2, ..., X_*^t)$ be the solution for (24--26). The multipliers w should be updated as follows:

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

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

In the next section, the selection of step size would be introduced.

3.2.3. Choosing step size

Algorithm 1 Algorithm of Lagrangian relaxation

Initialize: initial number of iteration $q = 0$, $flag = 0$, $\lambda = 1$, $w = 0$, 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 (29)-(31) and obtain $X_* = (X_*^1, ..., X_*^t)$, $L(\mu)$.
  if X_* is feasible:
    $UB = z(X_*)$
  endif
  if $L(\mu) < LB$:
    $flag = 3$
  else:
    if $L(\mu) - LB < eps * \max(1, LB)$:
      $flag = flag + 1$
    endif
    if $L(w) > LB$:
      $LB = L(\mu)$
    endif
  endif
  if $flag > 2$:
    $\lambda = \lambda/2$
    $flag = 0$
  endif
  $\theta = \frac{\lambda[UB - L(\mu)]}{\| \sum_{k=1}^{t} X_*^k - cap \|}$
  update w with Equation (35), $q = q + 1$.
endwhile
Output: The optimal solution for original MCFP is LB

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, $\theta^q = \frac{1}{q}$ is a simple choice. In this paper, we use the step size as follows:

$$\theta = \frac{\lambda[UB - L(\mu)]}{\mathrm{P}\sum_{k=1}^{t} X_*^k - cap\,\mathrm{P}} \tag{37}$$

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

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:

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

$$\text{subject to} \quad AX \geq b \tag{39}$$

$$AeqX = beq \tag{40}$$

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

Recently, a number of optimizers are 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* (GNU Linear Programming Kit) is a package for solving large-scale linear (LP), mixed integer (MIP), and related optimization problems [26].

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 close to math formulations instead of those standard forms for solvers [27].

3.3.3. GUROBI

GUROBI is a commercial optimization solver for linear (LP), quadratic (QP), mixed integer (MIP), and several related programs. Similarly to *GLPK* and *CVXPY*, it can support a series of program and modeling languages, such as C++, Java, Python, and so on. [28]

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, the linear program can be solved with this tool [29].

Note that Lagrangian sub-problem (29--31) for each commodity k is a shortest-path problem in fact. The only difference is that the cost vector c is add by Lagrangian multiplier μ. Thus, *Dijkstra* method for finding the shortest path from $O(k)$ to $D(k)$ can also be used in addition to the presented program solvers. The speeds of Dijkstra method and program solvers all depend on the size of network, so their properties will be compared together in 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 structures of these networks are shown in Table 5, Table 6, and Table 7. The parameters in the first line are explained as follows: n, m, t are the numbers of nodes, edges, and commodities, separately; p is the percentage of edges 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 5: Grid networks

Instances	n	m	p	t	ADG	ASPLC	ASPLA
grid1	25	80	26.67%	50	6.4	3.2	3.2
grid2	25	80	26.67%	100	6.4	3.3	3.2
grid3	100	360	7.27%	50	7.2	6.4	6.6
grid4	100	360	7.27%	100	7.2	6.5	6.6
grid5	225	840	3.33%	100	7.5	10.2	10.0
grid6	225	840	3.33%	200	7.5	10.4	10.0
grid7	400	1520	1.90%	400	7.6	13.5	13.3
grid8	625	2400	1.23%	500	7.7	17.2	16.6
grid9	625	2400	1.23%	1000	7.7	16.8	16.6

Nine *grid* networks are shown in Table 5 with n ranging from 25 to 625. With the increase of network size, the average degrees all keep small values. Thus, the average shortest path length for commodities (*ASPLC*) and all node pairs (*ASPLA*) are both larger and larger. It means that for a larger grid network, we need longer paths to connect two arbitrary nodes on average.

4.2. Planar networks

The properties of five *planar* networks are shown in Table 6. The average degrees always keep large values 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 that in *grid* networks. It means that more commodities with different OD pairs need to be transported.

Table 6: Planar networks

Instances	n	m	p	t	ADG	ASPLC	ASPLA
planar1	30	150	34.48%	92	10.0	2.7	2.5
planar2	50	250	20.41%	267	10.0	3.6	3.5
planar3	80	440	13.92%	543	11.0	4.0	3.9
planar4	100	532	10.75%	1085	10.6	4.5	4.6
planar5	150	850	7.61%	2239	11.3	5.2	5.1

4.3. Airport networks

In addition to two groups of networks above, several *airport* networks of different countries are also discussed. As shown in Table 7, the average degree is high and most OD pairs can be connected within paths with two edges (*ASPLC*< 2). It means that nodes in these networks are connected strongly which is quite different from *grid* and *planar* networks.

Table 7: Airport networks

Instances	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:

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

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

Table 8: Gaps of grid problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
grid1	0	0	0	0
grid2	0	0	0	0
grid3	0	0	0	0
grid4	0	0	0	0
grid5	0	0	0	0.056%
grid6	0	0	0	*
grid7	0	0	0	*
grid8	0	0	0	*
grid9	0	*	0	*

Table 9: Gaps of grid problems with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
grid1	0.013%	0.013%	0.013%	0.016%	0.013%
grid2	0.932%	0.357%	0.360%	1.124%	0.382%
grid3	0.012%	0.012%	0.007%	0.016%	0.009%
grid4	0.010%	0.016%	0.023%	0.010%	0.007%
grid5	0.019%	0.011%	0.013%	0.008%	*
grid6	0.074%	0.047%	0.080%	0.100%	*
grid7	0.013%	0.011%	0.009%	0.009%	*
grid8	0.110%	0.024%	0.038%	*	*
grid9	0.022%	*	*	*	*

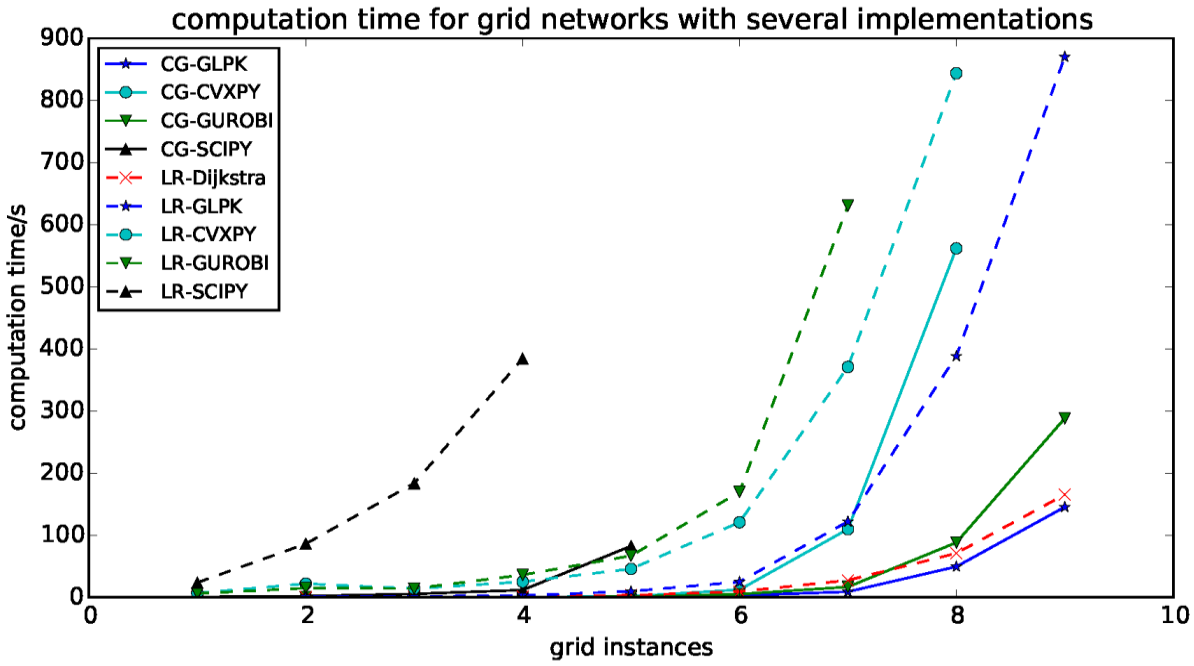

Figure 2: The comparison of computaion time for grid networks

The gaps of *grid* networks are shown in Table 8 and Table 9, where the symbol * represents the case that the running time is over 300s. Overall, the quality of column generation is better than that of Lagrangian relaxation. As shown in Table 8, the solutions obtained by column generation are all optimal except solving *grid5* with *SCIPY*. However, in Table 9, the solutions obtained by the five implementations for Lagrangian relaxation are within 1% of optimality in the case that they can get solutions.

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

The numbers of iterations are reported in Table 10 and Table 11. It is shown that, for each network, the numbers of iterations of column generation are much less than that of Lagrangian relaxation. This may be the major reason of the difference between their computation time.

Table 10: Numbers of iterations of grid problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
grid1	4	4	4	4
grid2	4	5	4	4
grid3	4	5	4	4
grid4	5	6	5	5
grid5	7	10	7	*
grid6	12	12	12	*
grid7	11	13	11	*
grid8	17	18	16	*
grid9	15	*	16	*

Table 11: Numbers of iterations of grid problems with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
grid1	44	44	44	54	44
grid2	78	73	70	76	79
grid3	50	50	55	48	50
grid4	53	53	48	54	50
grid5	53	52	52	55	*
grid6	72	64	68	66	*
grid7	62	64	64	68	*
grid8	76	76	74	*	*
grid9	87	84	*	*	*

5.2. Planar networks

Table 12: Gaps of planar problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
planar1	0	0	0	0
planar2	0	0	0	0
planar3	0	0	0	51%
planar4	0	0	0	*
planar5	0	0	0	*

Table 13: Gaps of grid problems with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
planar1	0.276%	0.276%	0.276%	0.276%	0.276%
planar2	0.324%	0.324%	0.324%	0.324%	0.324%
planar3	0.288%	0.288%	0.288%	0.288%	*
planar4	0.213%	0.166%	0.165%	0.213%	*
planar5	0.903%	0.903%	*	*	*

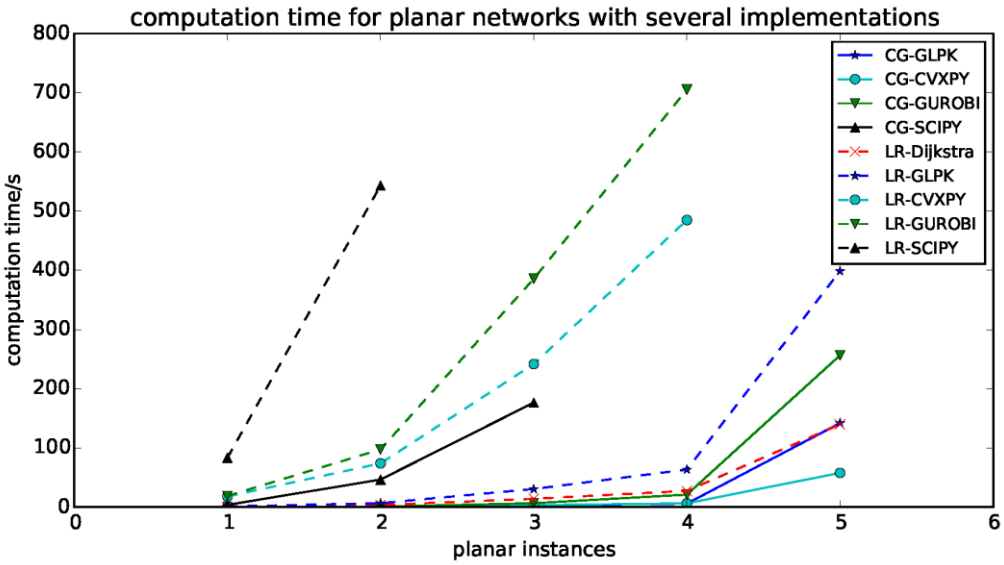

Figure 3: The comparison of computation time for planar networks

In Table 12 and Table 13, the gaps of *planar* networks are shown. Similarly to the results for *grid* networks, solutions obtained by column generation are all optimal except from *planar3* with *SCIPY*. Note that the gap of *planar3* with *SCIPY* in Table 12 is 51% and the solution in this case is wrong. This shows that *SCIPY* is not reliable while solving large-scale linear program because of its simplicity. The gaps with Lagrangian relaxation in Table 13 are all within 1% of optimality and are worse than column generation.

The computation time for *planar* networks is reported in Figure 3. The running 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 running time 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 numbers of iterations are shown in Table 14 and Table 15. Column generation needs much less iterations than Lagrangian relaxation, which is similar to the results of *grid* networks.

Table 14: Numbers of iterations of planar problems with column generation

Instance	GLPK	CVXPY	GUROBI	SCIPY
planar1	4	4	5	5
planar2	6	6	6	6
planar3	8	8	8	7
planar4	8	8	9	*
planar5	10	10	11	*

Table 15: Numbers of iterations of planar problems with Lagrangian relaxation

Instance	Dijkstra	GLPK	CVXPY	GUROBI	SCIPY
planar1	52	52	52	52	52
planar2	61	61	61	61	61
planar3	72	72	72	72	*
planar4	60	60	65	60	*
planar5	93	93	*	*	*

5.3. Airport networks

As shown in Table 16 and Table 17, the gaps of *airport* networks with column generation are smaller than that with Lagrangian relaxation in general. The quality of column generation is still better than Lagrangian relaxation.

Table 16: 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	*

Table 17: 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%	*	*	*

The computation time is shown in Figure 4. The labels *DE*, *FR*, *ES*, *AU*, *IN* on x-axes represent 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 18. Label * indicates that the running 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 18, 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.

Table 18: 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	*	*	*

Table 19: Numbers of iterations of airport networks 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	*

Table 20: Numbers of iterations of airport networks 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	*	*	*

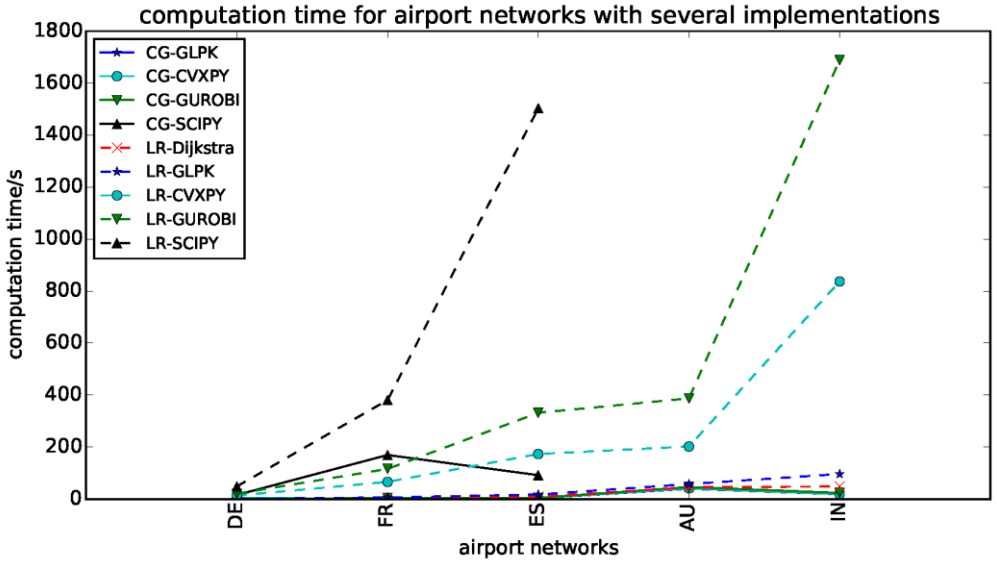

Figure 4: The comparison of computation time for airport networks

For column generation, with the increase of network size and commodity number, the speed of *CVXPY*

outperforms *GLPK* which is the fastest implementation. Finally, time of *CVXPY* is much shorter than *GLPK* for the *airport* network of *China* as shown in Table 18. Taken together with the results of *planar* networks, it is shown that *CVXPY* can have a good property while solving MCFP with a large number of commodities.

The numbers of iterations are shown in Table 19 and Table 20 and the result is similar to the other two groups of networks that column generation needs less iterations.

Table 21: Summary of implementations for column generation

Implementation	Gaps	Execution speed	Numbers of iterations
GLPK	0 if solution is obtained	fast	small
CVXPY	0 if solution is obtained	very fast for large commodity number	small but occasionally large
GUROBI	0 if solution is obtained	medium speed	small
SCIPY	large if network size is large	slowest	small

Table 22: 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

5.4. Summary

Column generation has better properties than Lagrangian relaxation for solving MCFP in general. However, in large networks with large number of commodities (like the *airport* network of *China*), the best performances of Lagrangian relaxation are close to the column generation. A summary of these implementations for the two algorithms is shown in Table 21 and Table 22.

6. Conclusion

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

In the evaluation, the objective function gaps, computation time, and numbers of iterations of 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 MCFP. However, for large networks with large 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 MCFP with a large number of commodities. For Lagrangian relaxation, it is shown that using Dijkstra shortest-path method to solve the Lagrangian sub-problem is the best choice. In general, *GUROBI* performs a medium level in both algorithms and *SCIPY* is always the worst one.

This paper lays out the baseline for evaluating implementations of MCFP algorithm. 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 further 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.

Acknowledgement

This study is supported by the Research Fund from National Natural Science Foundation of China (Grant No. 61521091, No. 61650110516 and No. 61601013).

References

1. Sun, X., Wandelt, S., Linke, F.. Temporal evolution analysis of the european air transportation system: air navigation route network and airport network. *Transportmetrica B* 2015;3(2):153–168.
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.. Network flows. Tech. Rep.; DTIC Document; 1988.

3. Tomlin, J.. Minimum-cost multicommodity network flows. *OPER RES* 1966;14(1):45–51.
4. Barnhart, C., Hane, C.A., Johnson, E.L., Sigismondi, G.. A column generation and partitioning approach for multi-commodity flow problems. *TELECOMMUN SYST* 1994;3(3):239–258.
5. Barnhart, C., Hane, C.A., Vance, P.H.. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *OPER RES* 2000;48(2):318–326.
6. Gondzio, J., Gonzalez-Brevis, P., Munari, P.. Large-scale optimization with the primal-dual column generation method. *Mathematical Programming Computation* 2013;:1–36.
7. Gondzio, J., Gonzalez-Brevis, P., Munari, P.. New developments in the primalcdual column generation technique. *EUR J OPER RES* 2013;224(1):41–51.
8. Gondzio, J., Gonzalez-Brevis, P.. A new warmstarting strategy for the primal-dual column generation method. *MATH PROGRAM* 2014;152:1–34.
9. Karakostas, G.. Faster approximation schemes for fractional multicommodity flow problems. *ACM T ALGORITHMS* 2008;4(1):13.
10. Garg, N., Konemann, J.. Faster and simpler algorithms for multicommodity flow and other fractional packing problems 1998;.
11. Fleischer, L.K.. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM J DISCRETE MATH* 2000;13(4):505–520.
12. Retvari, G., Biro, J.J., Cinkler, T.. A novel lagrangian-relaxation to the minimum cost multicommodity flow problem and its application to ospf traffic engineering. In: *International Symposium on Computers and Communications, 2004. Proceedings. ISCC*. 2004, p. 957–962 Vol.2.
13. Babonneau, F., Vial, J.P.. Solving large-scale linear multicommodity flow problems with an active set strategy and proximal-accpm. *OPER RES* 2006;54(1):184–197.
14. Moradi, S., Raith, A., Ehrgott, M.. A bi-objective column generation algorithm for the multi-commodity minimum cost flow problem. *EUR J OPER RES* 2015;244(2):369–378.
15. Bauguion, P.O., 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, P.O., Ben-Ameur, W., Gourdin, E.. Efficient algorithms for the maximum concurrent flow problem. *NETWORKS* 2015;65(1):56–67.
17. Cort´es, P., Mu˜nuzuri, J., Guadix, J., Onieva, L.. Optimal algorithm for the demand routing problem in multicommodity flow distribution networks with diversification constraints and concave costs. *INT J PROD ECON* 2013;146(1):313–324.
18. Zhang, B., Qi, L., Yao, E.. A multi-commodity production and distribution model in supply chain. In: *Supply Chain Management and Information Systems (SCMIS), 2010 8th International Conference on. IEEE;* 2010, 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. *TRANSPORT SCI* 2011;45(2):212–227.
20. Morabito, R., de Souza, M.C., Vazquez, M.. Approximate decomposition methods for the analysis of multicommodity flow routing in generalized queuing networks. *EUR J OPER RES* 2014;232(3):618–629.
21. Shitrit, H.B., Berclaz, J., Fleuret, F., Fua, P.. Multi-commodity network flow for tracking multiple people. *IEEE T PATTERN ANAL* 2014;36(8):1614–1627.
22. Wandelt, S., Sun, X.. Efficient compression of 4d-trajectory data in air traffic management. *IEEE T INTELL TRANSP* 2015;16(2):844–853.
23. Barnhart, C., Krishnan, N., Vance, P.H.. *Multicommodity Flow Problems.* Springer US; 2001.
24. Jordi, C., Jordi, C.. Improving an interior-point algorithm for multicommodity flows by quadratic regularizations. *NETWORKS* 2012;59(1):117–131.
25. 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.
26. Makhorin, A.. Glpk (gnu linear programming kit). 2008.
27. Diamond, S., Chu, E., Boyd, S.. Cvxpy: A python-embedded modeling language for convex optimization, version 0.2. 2014.
28. Optimization, G., et al. Gurobi optimizer reference manual. URL: http://www gurobi com 2012;.
29. Jones, E., Oliphant, T., Peterson, P., et al. Scipy: Open source scientific tools for python. URL http://www scipy org 2015;73:86.