Scientia Iranica E (2017) 24(1), 319–329

Sharif University of Technology
Scientia Iranica
Transactions E: Industrial Engineering
www.scientiairanica.com

Adjusting an infeasible network by minimizing the sum of the violation costs

M. Ghiyasvand*

Department of Mathematics, Faculty of Science, Bu-Ali Sina University, Hamedan, Iran.

Received 14 February 2015; received in revised form 24 November 2015; accepted 9 January 2016

KEYWORDS
Infeasible network;
The cost scaling algorithm;
The minimum cost flow problem;
Parallel arcs;
Convex cost.

Abstract. In this paper, for a given infeasible network, we change the lower and upper bounds, such that the sum of the violations costs from the lower and upper bounds is minimum. We call this problem as the adjusting problem and show that it is transformed to a minimum cost flow problem on a special parallel network. Thus, the adjusting problem is solved using minimum cost flow algorithms. Solving a minimum cost flow problem with parallel arcs, in practice, is complicated and needs more time in comparison with a minimum cost flow problem without parallel arcs. If the parallel arcs are eliminated, then, we achieve substantial saving in the storage requirements, which translates into enhanced speed of algorithms. One of the best algorithms to solve the minimum cost flow problem is the cost scaling algorithm of Goldberg and Tajan (1990). In this paper, we present two modified versions of their algorithm to solve the adjusting problem. In the first modification, in order to achieve an enhanced speed of algorithm, the parallel arcs are eliminated using an especial residual network. In the second modification, the adjusting problem is transformed to a convex cost flow problem, and the cost scaling algorithm is modified in a way which performs fewer operations than our first implementation.

1. Introduction

A wide variety of engineering and management problems involve optimization of network flows, that is, how objects move through a network. Examples include coordination of trucks in a transportation system, routing of packets in a communication network, and sequencing of legs for air travel. Such problems often involve few indivisible objects, and this leads to a finite set of feasible solutions. Li et al. [1] presented a good survey of network flow applications. Some recent applications of network flows problem have been presented in [2-10].

Let $D = (N, A)$ be a directed network with node set, N, arc set, A, lower bounds, l, and the upper bounds, u. Network D is feasible if there is a flow, x, satisfying in the following conditions:

Conservation:

$$\sum_{j \in N} x_{ij} - \sum_{j \in N} x_{ji} = 0, \qquad \text{for each } i \in N. \tag{1}$$

Boundedness:

$$l_{ij} \leq x_{ij} \leq u_{ij}, \qquad \text{for each } (i,j) \in A. \tag{2}$$

A classic method to diagnose the feasibility or infeasibility of D is to first choose any x', satisfying the boundedness, and compute excess e_i at each node i, defined by $e_i = \sum_{j \in N} x'_{ji} - \sum_{j \in N} x'_{ij}$. Define a source node s, and a sink node t. For every node i with $e_i > 0$, introduce an arc (s, i) with upper bound e_i, and for every node j with $e_j < 0$, introduce an arc (j, t) with upper bound $-e_j$. For each arc (i, j) in the original network, define two arcs (i, j) and (j, i) with upper bounds $u_{ij} - x'_{ij}$ and $x'_{ij} - l_{ij}$. All arcs in the

*. *Tel.: 081-38380987; Fax: 081-38380987*
E-mail address: mghiyasvand@basu.ac.ir

transformed network have a lower bound of zero. There exists a feasible flow for the original network if and only if the maximum flow from node s to node t in the transformed network saturates all arcs from s (to t) (see [11]). A faster algorithm to solve the maximum flow problem was presented by Orlin [12]. Other methods, for diagnosing feasibility or infeasibility of a network, were discussed in [13-31]. Some recent related results are presented in [32-35].

Infeasibility may be introduced into a network during formation, reformation, or when combining several smaller models into one large model (as is often done in econometrics). But, what should we do if the network D is infeasible? In this paper, we decrease some lower bounds and increase some upper bounds in order to achieve feasibility. Let b_{ij} be the cost of increasing u_{ij} or decreasing l_{ij} by one unit. It is very important that the bounds are relaxed, such that the sum of costs of relaxation is minimum. Denote the lower and upper relaxations of the bounds by p_{ij} and q_{ij}, respectively. Consider the following problem, which we call it *the adjusting problem*:

$$\min \quad \sum_{(i,j)\in A} b_{ij}(p_{ij}+q_{ij}),$$

$$\text{s.t.} \quad \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0, \quad \text{for each } i \in N,$$

$$\max(0, l_{ij}-p_{ij}) \le x_{ij} \le u_{ij}+q_{ij},$$

$$\text{for each } (i,j) \in A.$$

Note that $\max(0, l_{ij}-p_{ij})$ is used instead of $l_{ij}-p_{ij}$ in order to have nonnegative lower bounds. McCormick [16] considered a special case of this problem with $b_{ij} = 1$ for each $(i,j) \in A$. He called this special case as the minsum objective and presented some methods to solve it (see Pages 185-187 of [16]). In this paper, we consider the general case of b_{ij}s and show that the adjusting problem is solved by the minimum cost flow algorithms using a parallel network that has three parallel arcs between each pair node. Thus, the adjusting problem is transformed to a minimum cost flow problem on a special parallel network; we call this problem as *the PA-problem*. Therefore, each minimum cost flow algorithm solves the adjusting problem.

Let n, m, U, and C be the number of vertices, number of arcs, maximum arc capacity, and maximum absolute value of an arc cost, respectively. The best running times for the minimum cost flow problem are the $O((m\log U)(m+n\log n))$-time method of Edmonds and Karp [36], the $O(nm\log(n^2/m)\log(nC))$-time method of Goldberg and Tarjan [37], the $O((m\log n)(m+n\log n))$-time method of Orlin [38] and Vygen [39], and the $O(nm(\log\log U)\log(nC))$-time of Ahuja et al. [40]. Each of these algorithms is the best for a different range of parameters n, m, U, and C. Some recent papers on the minimum cost flow problem have been presented in [41-51].

Ahuja et al. [11] called the algorithm of Goldberg and Tarjan as the cost scaling algorithm. Solving a minimum cost flow problem with parallel arcs, in practice, is complicated and needs more time in comparison with a minimum cost flow problem without parallel arcs. In the first part of this paper, the cost scaling algorithm is modified for the PA-problem, in which there exist three arcs between each pair node; the number of arcs is $3m$. Although using $3m$ instead of m cannot improve the running time of the cost scaling algorithm, but if the parallel arcs are eliminated, then we achieve substantial saving in the storage requirements, which translates into enhanced speed of algorithms. We construct a special residual network that permits us to eliminate the parallel arcs.

In the second part of this paper, our second implementation of the cost scaling algorithm is presented. We convert the adjusting problem to a convex minimum cost flow problem and modify the cost scaling algorithm to solve it. We prove that our second implementation of the cost scaling algorithm performs fewer operations than our first implementation.

This paper consists of five sections, in addition to Introduction. Section 2 is a review of the cost scaling algorithm due to Goldberg and Tarjan [37]. In Section 3.1, the adjusting problem is transformed to a minimum cost flow problem, where there exist three parallel arcs between any pair of nodes. The adjusting problem is transformed to a convex cost flow problem in Section 3.2. In Section 4.1, our modification of the cost scaling algorithm to solve the adjusting problem on the parallel network is presented. Our implementation on the convex cost flow problem is described in Section 4.2. The comparison of our two implementations is presented in Section 5. Finally, Section 6 concludes the paper.

2. Goldberg and Tarjan's algorithm

Our algorithm requires some understanding of the linear cost scaling algorithm for minimum cost flow problem. We refer the readers to the paper by Goldberg and Tarjan [37] or the book of Ahuja et al. [11] for a description of this algorithm. We now briefly describe it using the notation given in [11]. The minimum cost flow problem is:

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij},$$

$$\text{s.t.} \quad \text{Conditions (1) and (2) are satisfied.}$$

A *pseudoflow* x is any function $x : A \to R$ that satisfies the boundedness on arc flows, but may violate the conservation constraints at nodes. For any pseudoflow x, the imbalance of node i is defined as:

$$e(i) = \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{j:(i,j)\in A\}} x_{ij}, \quad \text{for all } i \in N.$$

We refer to a pseudoflow x with $e(i) = 0$ for all $i \in N$ as a *flow*.

The cost scaling algorithm proceeds by constructing the residual network $G(x)$ defined as follows with respect to a pseudoflow x. For each $(i, j) \in A$, the residual network $G(x)$ contains two arcs: (i, j) and (j, i). The arc (i, j) has cost c_{ij} and residual capacity $r_{ij} = u_{ij} - x_{ij}$; the arc (j, i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = x_{ij}$.

The linear cost scaling algorithm maintains a value $\pi(i)$ for each node $i \in N$. We refer to the vector π as a vector of *node potentials*. For a given residual network, $G(x)$, and a set of node potentials, π, the reduced cost of an arc (i, j) is defined as $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$. A flow or a pseudoflow x is said to be ϵ-optimal for some $\epsilon > 0$ if x, together with some node potential vector π, satisfies the *ε-optimality conditions*: $c^{\pi}_{ij} \geq -\varepsilon$ for every arc (i, j) in $G(x)$.

Lemma 1 [37]. For a minimum cost flow problem with integer costs, any feasible flow is ϵ-optimal whenever $\epsilon \geq C$. Moreover, if $\epsilon < 1/n$, then any ϵ-optimal feasible flow is an optimal flow. ■

The algorithm treats ϵ as a parameter obtaining ϵ-optimal flows for successively smaller values of ϵ. Initially, $\epsilon = C$ and $\pi = 0$. The algorithm performs cost scaling phases by repeatedly applying an improve-approximation procedure that transforms an ϵ-optimal flow into an $\epsilon/2$-optimal flow.

Hence, by Lemma 1, after $1 + \log(nC)$ phases, the algorithm terminates with an optimal flow. Algorithm 1 shows the framework of the cost scaling algorithm.

The essential operation in each phase is the improve-approximation procedure that transforms an ϵ-optimal flow into an $\epsilon/2$-optimal flow. It does this by converting the input ϵ-optimal flow into a 0-optimal pseudoflow, and then converting the pseudoflow into a flow while always maintaining an $\epsilon/2$-optimal solution. For converting the ϵ-optimal flow into a 0-optimal pseudoflow, the improve-approximation procedure lets $x_{ij} = 0$ if $c^{\pi}_{ij} > 0$, and lets $x_{ij} = u_{ij}$ if $c^{\pi}_{ij} < 0$. An arc (i, j) in the residual network is called *admissible* if $-\epsilon/2 \leq c^{\pi}_{ij} \leq 0$, and a node i is called *active* if $e(i) > 0$.

For changing the 0-optimal flow into an $\epsilon/2$-optimal flow, the basic operation is to select an active node i and perform *pushes* on admissible arcs (i, j) emanating from node i. If $\delta < r_{ij}$, then the push is called *a nonsaturating push*, or else a *saturating push*. When the network contains no admissible arc, the algorithm updates the node potential $\pi(i)$ by *relabeling*. Algorithm 2 summarizes the improve-approximation procedure.

The improve-approximation procedure performs $O(n^2m)$ pushes and $O(n^2)$ labels [37]. Thus, the procedure runs in $O(n^2m)$ time, and the running time of the algorithm is $O(n^2m \log(nC))$. Goldberg and Tarjan [37] used the dynamic tree data structure to improve the procedure. The heart of this method is the procedure *send(i)*, which pushes flow along a *path* P from a nonroot active node i to the root of the tree containing the node i and repeats these steps until $e(i) = 0$ or node i is a tree root. The value of push on path P is computed by $\overline{\delta} = \min\{e(i), \min\{r_{ij} : (i, j) \in P\}\}$. The improve-approximation procedure using the dynamic tree structure is implemented in $O(nm \log(n^2/m))$ time [37]. Therefore, the cost scaling algorithm runs in $O(nm \log(n^2/m) \log(nC))$ time, which is one of the best running times to solve the minimum cost flow problem.

```
Begin
    π = 0 and ε = C;
    Let x be any feasible flow;
    While ε ≥ 1/n do
    Begin
        Improve-approximation;
        ε = ε/2;
    End;
End.
```

Algorithm 1. Framework of Goldberg and Tarjan's algorithm.

3. Transformations of the adjusting problem

3.1. Solving the adjusting problem using min-cost flow algorithms

In this section, the adjusting problem is transformed to a min-cost flow problem using a parallel network. Supposing that an infeasible network is given, we

```
Begin
    For every (i, j) ∈ A do
        If c^π_ij > 0 then x_ij = 0;
        Else if c^π_ij < 0 then x_ij = u_ij;
    Compute node imbalances;
    While the network contains an active node do
    Begin
        Select an active node i;
        If G(x) contains an admissible arc (i, j) then;
            Push δ = min{e(i), r_ij} units of flow from node i
            to node j;
        Else π(i) = π(i) + ε/2;
    End;
End.
```

Algorithm 2. Procedure improve-approximation.

replace each arc by three parallel arcs with upper bounds l_{ij}, $u_{ij}-l_{ij}$, ∞, lower bounds 0, 0, 0, and costs $-b_{ij}$, 0, b_{ij}, respectively. The new network is a *parallel network*, which each arc (i,j) on the original network is replaced with three arcs $(i,j)^1$, $(i,j)^2$, and $(i,j)^3$ on the parallel network (see Figure 1). Consider the minimum cost flow problem on the parallel network; we call it *the parallel adjusting problem* or *the PA-problem*.

Theorem 1. The adjusting problem is solved using each solution of the PA-problem.

Proof. Let x^* be an optimal flow for the PA-problem and $x^*_{ij}(1)$, $x^*_{ij}(2)$, and $x^*_{ij}(3)$ be the flows on arcs $(i,j)^1$, $(i,j)^2$, and $(i,j)^3$, respectively. For each arc (i,j), define:

$$x_{ij} = x^*_{ij}(1) + x^*_{ij}(2) + x^*_{ij}(3). \tag{3}$$

By Eq. (3), Figure 1, and the definition of b_{ij}, we yield the sum of the violations costs from the lower and upper bounds in the original network with respect to x is minimum. ■

Therefore, for solving the adjusting problem, it is enough that we solve a minimum cost flow problem on the parallel network. Sometimes, the values of the adjustment costs in the adjusting and PA problems are different, but they conclude the same feasible flow. For example, consider the given network in Figure 2. This network is infeasible and can be feasible by lowering the capacity lower bound of (y,z) from 3 to 2. So, the adjustment cost is 1. In the corresponding PA-problem, we have arcs $(x,y)^1$ and $(z,x)^1$ with (cost, lower, upper) = (-1,0,1), arc $(y,z)^1$ with (cost, lower, upper) = (-1,0,3), arcs $(x,y)^2$, $(y,z)^2$, and $(z,x)^2$ with (cost, lower, upper) = (0,0,1), and arcs $(x,y)^3$, $(y,z)^3$, and $(z,x)^3$ with (cost,lower,upper) = $(1,0,\infty)$. Then, we get a feasible flow by assigning flow of 1 unit on arcs $(x,y)^1$, $(x,y)^2$, $(z,x)^1$, and $(z,x)^2$, and 2 units on $(y,z)^1$. The total cost is -4, which is minimum. Even the adjustment costs are different, but both of them present a feasible network by lowering the capacity of the lower bound of (y,z) from 3 to 2.

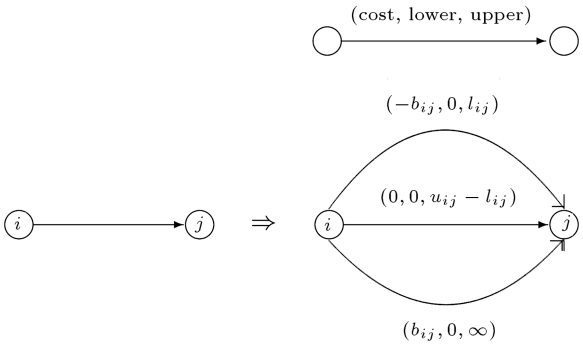

Figure 1. The arcs between a pair of nodes in the parallel network.

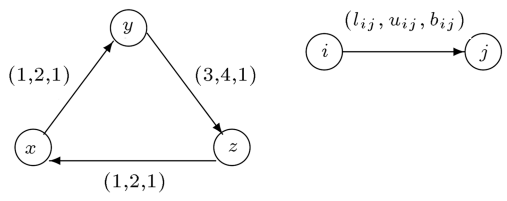

Figure 2. An example network.

3.2. Transformation of the adjusting problem as a convex cost flow problem

In this section, we look at the adjusting problem in another way in which there is no bound on the arcs, but the cost of flow is the following convex function:

$$C_{ij}(x_{ij}) = \begin{cases} -b_{ij}, & \text{if } x_{ij} < l_{ij}, \\ 0, & \text{if } l_{ij} \le x_{ij} < u_{ij}, \\ b_{ij}, & \text{if } x_{ij} \ge u_{ij}. \end{cases}$$

Thus, the adjusting problem is transformed to:

$$\min \quad \sum_{(i,j)\in A} C_{ij}(x_{ij}),$$

$$\text{s.t.} \quad \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0, \qquad i \in N.$$

Although there are no lower and upper constrains on the variables x_{ij}, the next lemma shows that we can bound it.

Lemma 2. For changing an infeasible network into a feasible network, the maximum relaxation in a lower bound or upper bound is nmU.

Proof. Consider an infeasible network $D = (N, A)$. In phase I max flow, the maximum flow from source to sink cannot saturate all the arcs of the source. By the phase I max flow, the sum of capacities of all arcs of the source is $\sum_{e_i>0} e_i$. Consider arc $(i,j) \in A$ in the worst case for saturating all arcs of the source, we relax l_{ij} or u_{ij} by $\sum_{e_i>0} e_i$. Therefore, by $\sum_{e_i>0} e_i \le nmU$, the maximum increase in u_{ij} or the maximum decrease in l_{ij} is nmU. ■

By Lemma 2, we conclude the following problem:

$$\min \quad \sum_{(i,j)\in A} C_{ij}(x_{ij}),$$

$$\text{s.t.} \quad \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = 0, \qquad \text{for each } i \in N,$$

$$k_{ij} \le x_{ij} \le u_{ij} + M, \qquad \text{for each } (i,j) \in A,$$

where $M = nmU$ and $k_{ij} = \max\{0, l_{ij} - M\}$. We call this problem *the convex-adjusting problem* or *the CA-problem*. Note that k_{ij} is used instead of $l_{ij} - M$ in order to have a nonnegative lower bound.

4. Solving the parallel adjusting and convex adjusting problems using the cost scaling algorithm

4.1. Describing Goldberg and Tarjan's algorithm for the parallel adjusting problem

By Theorem 1, the adjusting problem is solved using minimum cost flow algorithms on the PA-problem. In fact, we can consider the adjusting problem as a minimum cost flow problem by replacing each arc (i, j) with three parallel arcs $(i, j)^1$, $(i, j)^2$, and $(i, j)^3$. As mentioned in Section 2, the cost scaling algorithm maintains a residual network at every step. The parallel and residual networks have parallel arcs between any pair of nodes. For example, consider a flow of 5 units in arc (i, j) of Figure 3(a). The resulting flow in the parallel network has 3 units on $(i, j)^1$, 2 units on $(i, j)^2$, and zero units on $(i, j)^3$ (see Figure 3(b)). The definition of the residual network in Section 2 implies that the residual network contains arcs $(j, i)^1$, $(j, i)^2$, $(i, j)^2$, and $(i, j)^3$ (see Figure 3(c)).

The solution of the PA-problem satisfies the property that if the flow on arc $(i, j)^k$ is positive, the flow on each of the arc $(i, j)^1, \cdots, (i, j)^{k-1}$ equals the arc's capacity, and if the flow on arc $(i, j)^k$ is strictly less than its upper bound, the flow on each of the arcs $(i, j)^{k+1}, \cdots, (i, j)^3$ is zero. Thus, each solution of the PA-problem is called as a contiguous solution (see [11]). The contiguity of the solution implies that if we wish to send additional flow from node i to node j, we will send it through the arc $(i, j)^2$, and if we wish to send flow from node j to node i, we send it through the arc $(j, i)^2$. This observation implies that we do not need to maintain other arcs between this pair of nodes in the residual network; maintaining just two arcs $(i, j)^2$ and $(j, i)^2$ is suffcient, because those are the arcs that matter at this point. Eliminating parallel arcs permits us to achieve substantial saving in the storage requirements, which translates into enhanced speed of algorithm (for more information see [11]).

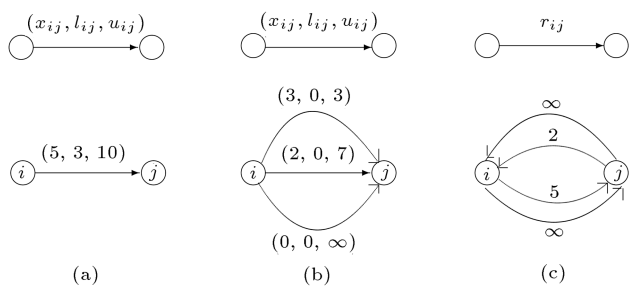

Figure 3. (a) An (i, j) in original network. (b) Flows on parallel arcs in the parallel network corresponding to (i, j). (c) Arcs in the residual network.

By the preceding discussion, we construct *the condensed residual network* by the following method. For each arc $(i, j) \in A$, the residual network contains two arcs (i, j) and (j, i) with the following costs and residual capacities:

If $x_{ij} < l_{ij}$, then $r_{ij} = l_{ij} - x_{ij}$, $c_{ij} = -b_{ij}$, $r_{ji} = x_{ij} - k_{ij}$ and $c_{ji} = b_{ij}$.

If $x_{ij} = l_{ij}$, then $r_{ij} = u_{ij} - l_{ij}$, $c_{ij} = 0$, $r_{ji} = k_{ij} - l_{ij}$ and $c_{ji} = b_{ij}$.

If $l_{ij} < x_{ij} < u_{ij}$, then $r_{ij} = u_{ij} - x_{ij}$, $c_{ij} = 0$, $r_{ji} = x_{ij} - l_{ij}$ and $c_{ji} = 0$.

If $x_{ij} = u_{ij}$, then $r_{ij} = M$, $c_{ij} = b_{ij}$, $r_{ji} = u_{ij} - l_{ij}$ and $c_{ji} = 0$.

If $x_{ij} > u_{ij}$, then $r_{ij} = u_{ij} + M - x_{ij}$, $c_{ij} = b_{ij}$, $r_{ji} = x_{ij} - u_{ij}$ and $c_{ji} = -b_{ij}$.

For each arc (i, j) in the condensed residual network, the residual capacity is the maximum of flow change, such that the flow cost remains consist. For example, in Figure 3, we have $3 = l_{ij} < x_{ij} = 5 < u_{ij} = 10$, so, we have $r_{ij} = 5$, $c_{ij} = 0$, $r_{ji} = 2$, and $c_{ji} = 0$.

The cost scaling algorithm for the parallel network is exactly the same as the algorithm discussed in Section 2. But, we construct the condensed residual network in a different way. After each push, updating the residual network might add some arcs to the condensed residual network, delete some others, and change the costs of some arcs remained in the residual network. In the dynamic tree structure of the cost scaling algorithm, we use r_{ij}s of the condensed residual network to compute value $\overline{\delta}$. After sending $\overline{\delta}$ units of flow on the path P, we update values of r_{ij}, r_{ji}, c_{ij}, and c_{ji} for each arc (i, j) using the definition of the condensed residual network. In the condensed residual network, we do not need to have the parallel arcs, so, we achieve substantial saving in the storage requirements. Hence, the condensed residual network helps to get an enhanced speed of the algorithm to solve the PA-problem.

4.2. Modifying Goldberg and Tarjan's algorithm for the convex adjusting problem

In Section 3.2, we showed that the adjusting problem is solved using the CA-problem. In this section, the cost scaling algorithm is modified to solve the CA-problem. For a given arc flow x, we construct the residual network $G(x)$ as follows: for each $(i, j) \in A$, the residual network $G(x)$ contains two arcs (i, j) and (j, i) with costs $c_{ij}(x)$ and $c_{ji}(x)$, respectively. The values $c_{ij}(x)$ and $c_{ji}(x)$ depend on the value of x_{ij}:

If $x_{ij} < l_{ij}$, then $c_{ij}(x) = -b_{ij}$ and $c_{ji}(x) = b_{ij}$.
If $x_{ij} = l_{ij}$, then $c_{ij}(x) = 0$ and $c_{ji}(x) = b_{ij}$.
If $l_{ij} < x_{ij} < u_{ij}$, then $c_{ij}(x) = c_{ji}(x) = 0$.
If $x_{ij} = u_{ij}$, then $c_{ij}(x) = b_{ij}$ and $c_{ji}(x) = 0$.
If $x_{ij} > u_{ij}$, then $c_{ij}(x) = b_{ij}$ and $c_{ji}(x) = -b_{ij}$.

As Section 2, the reduced cost of an arc, $(i, j) \in G(x)$, is defined by $c^{\pi}_{ij}(x) = c_{ij}(x) - \pi_i + \pi_j$, and arc (i, j) is called an *admissible arc* if $-\epsilon/2 \leq c^{\pi}_{ij}(x) < 0$. An arc $(i, j) \in G(x)$ is called a *forward arc* if $(i, j) \in A$ and is called a *backward arc* if $(j, i) \in A$. For modifying the cost scaling algorithm, we need the following lemmas.

Lemma 3. In $G(x)$, the reduced costs of arcs (i, j) and (j, i) are not satisfied in the case of $c^{\pi}_{ij}(x) < 0$ and $c^{\pi}_{ji}(x) < 0$.

Proof. Supposing that for a pair (i, j) and (j, i) in $G(x)$, we have $c^{\pi}_{ij}(x) < 0$ and $c^{\pi}_{ji}(x) < 0$, which means $c_{ij}(x) - \pi_i + \pi_j < 0$ and $c_{ji}(x) - \pi_j + \pi_i < 0$. Thus, we get:

$$c_{ij}(x) + c_{ji}(x) < 0. \tag{4}$$

We show that there does not exist x_{ij} to satisfy Relation (4). If $x_{ij} < l_{ij}$, $x_{ij} > u_{ij}$, or $l_{ij} < x_{ij} < u_{ij}$, then $c_{ij}(x) = -c_{ji}(x)$, which contradicts Relation (4). If $x_{ij} = l_{ij}$, then $c_{ij}(x) = 0$ and $c_{ji}(x) = b_{ij}$. Also, if $x_{ij} = u_{ij}$, then $c_{ij}(x) = b_{ij}$ and $c_{ji}(x) = 0$. Hence, if $x_{ij} = l_{ij}$ or $x_{ij} = u_{ij}$, then $c_{ij}(x) + c_{ji}(x) \geq 0$, which contradicts Relation (4). ■

Lemma 3 says, for each $(i, j) \in A$, at most one of arcs (i, j) or (j, i) in $G(x)$ violates the 0-optimality condition. Therefore, if a forward (resp. backward) arc (i, j) violates 0-optimality condition, then we conclude Figure 4 (resp. Figure 5). The next lemmas show how 0-optimality condition is satisfied when a forward or backward arc (i, j) violates 0-optimality condition.

Lemma 4. Let (i, j) be a forward arc which violates 0-optimality conditions. f_{ij} is defined as shown in Box I.

If we send f_{ij} units of flow on (i, j), then (i, j) and (j, i) are satisfied in 0-optimality condition.

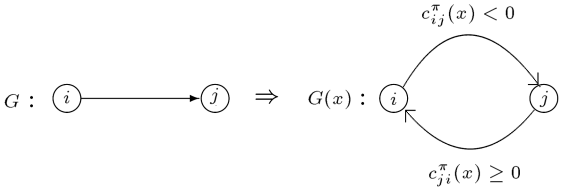

Figure 4. A forward arc (i, j) which violates 0-optimality condition.

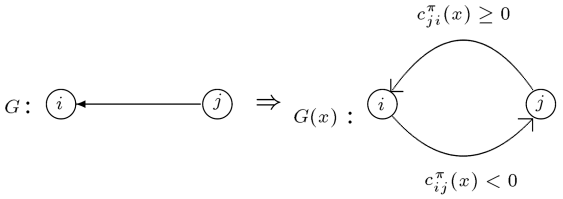

Figure 5. A backward arc (i, j) which violates 0-optimality condition.

Proof. By Lemma 3, we have $c^{\pi}_{ij}(x) < 0$ and $c^{\pi}_{ji}(x) \geq 0$ (see Figure 4). Consider the following cases:

a) $k_{ij} \leq x_{ij} < l_{ij}$. In this case, we have $c_{ij}(x) = -b_{ij}$, so, by $c^{\pi}_{ij}(x) < 0$, we get $-b_{ij} - \pi_i + \pi_j < 0$, which means:

$$-\pi_i + \pi_j < b_{ij}. \tag{5}$$

a.1) If $-\pi_i + \pi_j \geq 0$, then by increasing x_{ij} to l_{ij}, we get $c_{ij}(x) = 0$ and $c_{ji}(x) = b_{ij}$. Thus, $c^{\pi}_{ij}(x) = 0 - \pi_i + \pi_j \geq 0$, which means that by Relation (5), $c^{\pi}_{ji}(x) = b_{ij} - \pi_j + \pi_i > 0$;

a.2) If $-b_{ij} \leq -\pi_i + \pi_j < 0$, then by increasing x_{ij} to u_{ij}, we have $c_{ij}(x) = b_{ij}$ and $c_{ji}(x) = 0$, which means $c^{\pi}_{ij}(x) = b_{ij} - \pi_i + \pi_j \geq 0$ and $c^{\pi}_{ji}(x) = 0 - \pi_j + \pi_i > 0$;

a.3) If $-\pi_i + \pi_j < -b_{ij}$, then by increasing x_{ij} to $u_{ij} + M$, we get $c_{ji}(x) = -b_{ij}$. Hence, $c^{\pi}_{ji}(x) = -b_{ij} - \pi_j + \pi_i > 0$ (note that if $x_{ij} = u_{ij} + M$, then $G(x)$ does not have arc (i, j)).

b) $l_{ij} \leq x_{ij} < u_{ij}$. In this case, we have $c_{ij}(x) = 0$. Thus, by $c^{\pi}_{ij}(x) < 0$, we get:

$$-\pi_i + \pi_j < 0. \tag{6}$$

b.1) If $-\pi_i + \pi_j < -b_{ij}$, then by increasing x_{ij} to

$$f_{ij} = \begin{cases} l_{ij} - x_{ij}, & \text{if } k_{ij} \leq x_{ij} < l_{ij},\ -\pi_i + \pi_j \geq 0, & (\alpha - 1) \\ u_{ij} - x_{ij}, & \text{if } k_{ij} \leq x_{ij} < l_{ij},\ 0 > -\pi_i + \pi_j \geq -b_{ij}, & (\alpha - 2) \\ u_{ij} + M - x_{ij}, & \text{if } k_{ij} \leq x_{ij} < l_{ij},\ -\pi_i + \pi_j < -b_{ij}, & (\alpha - 3) \\ u_{ij} + M - x_{ij}, & \text{if } l_{ij} \leq x_{ij} < u_{ij},\ -\pi_i + \pi_j < -b_{ij}, & (\alpha - 4) \\ u_{ij} - x_{ij}, & \text{if } l_{ij} \leq x_{ij} < u_{ij},\ -\pi_i + \pi_j \geq -b_{ij}, & (\alpha - 5) \\ u_{ij} + M - x_{ij}, & \text{if } u_{ij} \leq x_{ij} < u_{ij} + M. & (\alpha - 6) \end{cases}$$

Box I

$u_{ij}+M$, we have $c_{ji}(x) = -b_{ij}$, which means $c^{\pi}_{ji}(x) = -b_{ij} - \pi_j + \pi_i > 0$;

b.2) If $-\pi_i + \pi_j \geq -b_{ij}$, then by increasing x_{ij} to u_{ij}, we get $c_{ij}(x) = b_{ij}$ and $c_{ji}(x) = 0$. Hence, $c^{\pi}_{ij}(x) = b_{ij} - \pi_i + \pi_j \geq 0$, which means that by Relation (6), $c^{\pi}_{ji}(x) = 0 - \pi_j + \pi_i > 0$.

c) $u_{ij} \leq x_{ij} < u_{ij}+M$. In this case, we have $c_{ij}(x) = b_{ij}$. So, by $c^{\pi}_{ij}(x) < 0$, $b_{ij} - \pi_i + \pi_j < 0$. Thus, by increasing x_{ij} to $u_{ij}+M$, we get $c_{ji}(x) = -b_{ij}$ and $c^{\pi}_{ji}(x) = -b_{ij} - \pi_j + \pi_i > 0$. ■

Lemma 5. Let (i,j) be a backward arc which violates 0-optimality conditions. f_{ij} is defined as shown in Box II. If we send f_{ij} units of flow on (i,j), then (i,j) and (j,i) are satisfied in the 0-optimality condition.

Proof. By Lemma 3, we have $c^{\pi}_{ij}(x) < 0$ and $c^{\pi}_{ji}(x) \geq 0$ (see Figure 5). Also, by sending f_{ij} units of flow on arc (i,j) in $G(x)$, the value of x_{ji} is decreased by f_{ij} on the arc (j,i) in the original network. Consider the following cases:

a) $k_{ji} < x_{ji} \leq l_{ji}$. In this case, we have $c_{ij}(x) = b_{ji}$. Thus, by $c^{\pi}_{ij}(x) < 0$, we get $b_{ji} - \pi_i + \pi_j < 0$. By decreasing x_{ji} to k_{ji}, we have $c_{ji}(x) = -b_{ji}$. Hence, $c^{\pi}_{ji}(x) = -b_{ji} - \pi_j + \pi_i > 0$ (note that if $x_{ji} = k_{ji}$, then there does not exist arc (i,j) in $G(x)$);

b) $l_{ji} < x_{ji} \leq u_{ji}$. In this case, we have $c_{ij}(x) = 0$. Hence, by $c^{\pi}_{ij}(x) < 0$, we get $0 - \pi_i + \pi_j < 0$, which means:

$$-\pi_j + \pi_i > 0. \tag{7}$$

b.1) If $-\pi_j + \pi_i < b_{ji}$, then by decreasing x_{ji} to l_{ji}, we get $c_{ij}(x) = b_{ji}$ and $c_{ji}(x) = 0$. Hence, $c^{\pi}_{ij}(x) = b_{ji} - \pi_i + \pi_j > 0$, which means that by Relation (7), $c^{\pi}_{ji}(x) = 0 - \pi_j + \pi_i > 0$;

b.2) If $-\pi_j + \pi_i \geq b_{ji}$, then by increasing x_{ji} to k_{ji}, we have $c_{ji}(x) = -b_{ji}$, so $c^{\pi}_{ji}(x) = -b_{ji} - \pi_j + \pi_i \geq 0$.

c) $u_{ji} < x_{ji} \leq u_{ji}+M$. In this case, we have $c_{ij}(x) = -b_{ji}$. So, by $c^{\pi}_{ij}(x) < 0$, we get $-b_{ji} - \pi_i + \pi_j < 0$, or:

$$-\pi_j + \pi_i > -b_{ji}. \tag{8}$$

c.1) If $-\pi_j + \pi_i \leq 0$, then by decreasing x_{ji} to u_{ji}, we get $c_{ij}(x) = 0$ and $c_{ji}(x) = b_{ji}$. Thus, $c^{\pi}_{ij}(x) = 0 - \pi_i + \pi_j \geq 0$, so, by Relation (8), $c^{\pi}_{ji}(x) = b_{ji} - \pi_j + \pi_i > 0$;

c.2) If $0 < -\pi_j + \pi_i \leq b_{ji}$, then by decreasing x_{ji} to l_{ji}, we have $c_{ji}(x) = 0$ and $c_{ij}(x) = b_{ji}$. Hence, $c^{\pi}_{ji}(x) = 0 - \pi_j + \pi_i > 0$ and $c^{\pi}_{ij}(x) = b_{ji} - \pi_i + \pi_j \geq 0$;

c.3) If $-\pi_j + \pi_i > b_{ji}$, then, by decreasing x_{ji} to k_{ij}, we get $c_{ji}(x) = -b_{ji}$, which means $c^{\pi}_{ji}(x) = -b_{ji} - \pi_j + \pi_i > 0$. ■

We use Lemmas 4 and 5 to modify the cost scaling algorithm in order to solve the CA-problem. Algorithm 3 shows the method, which uses f_{ij}s instead of r_{ij}s. The next lemma says that the procedure of Algorithm 3 computes an $\epsilon/2$-optimal flow.

Lemma 6. The improve-approximation procedure always maintains the $\epsilon/2$-optimality condition.

Proof. We use induction on the number of pushes and relabels. At the beginning of the procedure, for each arc $(i,j) \in G(x)$ such that $c^{\pi}_{ij} < 0$, flow is sent on arc (i,j) using Lemmas 4 and 5. Thus, at this point, for each $(i,j) \in G(x)$, we have $c^{\pi}_{ij} \geq 0$, which means $c^{\pi}_{ij} \geq -\epsilon/2$. Hence, at the beginning of the procedure, we have a pseudoflow that is $\epsilon/2$-optimal. By Lemmas 4 and 5, each saturating push on an arc (i,j) holds the induction hypothesis. Now, consider a non-saturating push on an arc (i,j). By the procedure, we do the push operation on admissible arcs, so $-\epsilon/2 \leq c^{\pi}_{ij} < 0$. Thus, by Lemma 3, we get $c^{\pi}_{ji} > 0$. The cost of CA-problem is a convex function. So, by sending flow on the (i,j), the reduced cost of the arc (i,j) is increased (from a number which is at least $-\epsilon/2$), and the reduced cost of arc (j,i) is decreased. Lemmas 4 and 5 conclude that by sending f_{ij} amount of flow, both c^{π}_{ij} and c^{π}_{ji} become nonnegative. Hence, by sending a flow less than f_{ij}, the reduced cost of arc (i,j) remains between $-\epsilon/2$ and zero, and the reduced cost of arc (j,i) remains nonnegative. Therefore, a non-saturating push holds the induction hypothesis. Also,

$$f_{ij} = \begin{cases} x_{ji} - k_{ji}, & \text{if } k_{ji} < x_{ji} \leq l_{ji}, & (\beta-1) \\ x_{ji} - l_{ji}, & \text{if } l_{ji} < x_{ji} \leq u_{ji},\ -\pi_j + \pi_i < b_{ji}, & (\beta-2) \\ x_{ji} - k_{ji}, & \text{if } l_{ji} < x_{ji} \leq u_{ji},\ -\pi_j + \pi_i \geq b_{ji}, & (\beta-3) \\ x_{ji} - u_{ji}, & \text{if } u_{ji} < x_{ji} \leq u_{ji}+M,\ -\pi_j + \pi_i \leq 0, & (\beta-4) \\ x_{ji} - l_{ji}, & \text{if } u_{ji} < x_{ji} \leq u_{ji}+M,\ 0 < -\pi_j + \pi_i \leq b_{ji}, & (\beta-5) \\ x_{ji} - k_{ji}, & \text{if } u_{ji} < x_{ji} \leq u_{ji}+M,\ -\pi_j + \pi_i > b_{ji}. & (\beta-6) \end{cases}$$

Box II

```
Begin
    For every (i, j) ∈ G(x) s.t. c^π_ij < 0 do
        If (i, j) is a forward arc then, compute f_ij and
        send f_ij amount of flow on arc (i, j) using
        (α − 1), (α − 2), ··· , (α − 6);
        If (i, j) is a backward arc then, compute f_ji and
        send f_ji amount of flow on arc (j, i) using
        (β − 1), (β − 2), ··· , (β − 6);
    End for;
    Compute node imbalances;
    While the network contains an active node do
    Begin
        Select an active node i;
        If G(x) contains an admissible arc (i, j) then
            If (i, j) is forward, then, compute f_ij using
            (α − 1), (α − 2), ··· , (α − 6) and push δ = min{e(i), f_ij}
            units of flow from node i to node j;
            If (i, j) is backward, then, compute f_ij using
            (β − 1), (β − 2), ··· , (β − 6) and push δ = min{e(i), f_ij}
            units of flow from node i to node j;
        Else π(i) = π(i) + ε/2;
    End;
End.
```

Algorithm 3. Procedure improve-approximation for the CA-problem.

a relabel operation holds the induction hypothesis (see Section 10.3 of [11]).

Our improve-approximation procedure is similar to Section 2, but the residual capacities of f_{ij}s are used instead of those of r_{ij}s. Thus, all arguments of the running time analysis are similar to the cost scaling algorithm, which runs in $O(n^2 m \log(nC))$ time. In the dynamic tree structure version of the cost scaling algorithm, instead of r_{ij}s, we use f_{ij}s to compute $\overline{\delta}$. For each arc (i, j), both r_{ij} and f_{ij} are computed in $O(1)$; therefore, it runs in $O(nm \log(n^2/m))$ time.

5. Comparison of the algorithms presented in Sections 4.1 and 4.2

Two algorithms presented in Sections 4.1 and 4.2 differ only on f_{ij}s and r_{ij}s; Section 4.2 uses f_{ij}s, but Section 4.1 uses the residual capacities r_{ij}. These values are used only in push operations. In this section, we show that the implementation of Section 4.2 performs fewer operations than the implementation of Section 4.1.

Lemma 7. For each $(i, j) \in G(x)$: $f_{ij} \geq r_{ij}$.

Proof. First, consider the case that arc (i, j) is forward. If $x_{ij} < l_{ij}$, then $r_{ij} = l_{ij} - x_{ij}$, so, by $(\alpha - 1)$, $(\alpha - 2)$ and $(\alpha - 3)$, we have $f_{ij} \geq l_{ij} - x_{ij}$. If $l_{ij} \leq x_{ij} < u_{ij}$, then $r_{ij} = u_{ij} - x_{ij}$, thus, by $(\alpha - 4)$ and $(\alpha - 5)$, we get $f_{ij} \geq u_{ij} - x_{ij}$. If $x_{ij} \geq u_{ij}$, then $r_{ij} = u_{ij} + M - x_{ij} = f_{ij}$. Hence, if (i, j) is forward, then $f_{ij} \geq r_{ij}$.

Now, consider the case that (i, j) is backward. If $x_{ji} < l_{ji}$, then $r_{ij} = x_{ji} - k_{ji} = f_{ij}$. If $l_{ji} < x_{ji} \leq u_{ji}$, then $r_{ij} = x_{ji} - l_{ji}$; so, by $(\beta - 2)$ and $(\beta - 3)$, we have $f_{ij} \geq x_{ji} - l_{ji}$. If $x_{ji} > u_{ji}$, then $r_{ij} = x_{ji} - u_{ji}$; so, by $(\beta - 4)$, $(\beta - 5)$, and $(\beta - 6)$, we get $f_{ij} \geq x_{ji} - u_{ji}$. Therefore, if (i, j) is backward, then $f_{ij} \geq r_{ij}$. ■

By the proof of Lemma 7, in cases of $(\alpha - 2)$, $(\alpha - 3)$, $(\alpha - 4)$, $(\beta - 3)$, $(\beta - 5)$, and $(\beta - 6)$, we have $f_{ij} > r_{ij}$.

Lemma 8. The number of pushes in the algorithm of Section 4.2 is less than or equal to the number of pushes in the algorithm of Section 4.1.

Proof. First, we prove the claim for the cost scaling algorithm without the dynamic tree structure. The residual capacities r_{ij} and f_{ij} are used only in push operations. The value of each push on each arc (i, j) in Section 4.1 is $A = \min\{e(i), r_{ij}\}$ and it is $B = \min\{e(i), f_{ij}\}$ in Section 4.2. By Lemma 7, we yield $B \geq A$. Thus, for transforming active nodes to nonactive nodes, the algorithm with values f_{ij}s is faster than the algorithm with values r_{ij}s. Therefore, the number of pushes using f_{ij}s is less than or equal to the number of pushes using r_{ij}s. ■

Theorem 2. The implementation of the cost scaling algorithm presented in Section 4.2 performs fewer operations than the implementation presented in Section 4.1.

Proof. The two implementations presented in Sections 4.1 and 4.2 differ only on f_{ij}s and r_{ij}s. The values f_{ij} and r_{ij} are used only in push operations. Also, active nodes are transformed into nonactive

nodes using push operations. Thus, by Lemma 8, the implementation presented in Section 4.2 transforms active nodes into nonactive nodes faster than the implementation presented in Section 4.1. Therefore, the implementation presented in Section 4.2 performs fewer push operations than the implementation presented in Section 4.1. Other operations of the two implementations are the same, thus the claim is concluded.

6. Conclusion

This paper presents the general case of the minsum objective defined by McCormick [16]. We call the problem as the adjusting problem and transform it to a minimum cost flow problem in a parallel network. The cost scaling algorithm has one of the best running times to solve the minimum cost flow problem for a range of parameters n, m, U, and C. Solving a minimum cost flow problem with parallel arcs, in practice, is complicated and needs more time in comparison with a minimum cost flow problem without parallel arcs. If parallel arcs are eliminated, then we achieve substantial saving in the storage requirements, which translates into enhanced speed of algorithms. We first modified the cost scaling algorithm to the parallel network by defining the condensed residual network, which concludes an enhanced speed of the algorithm. Then, the adjusting problem is transformed to a convex cost flow problem and modified the cost scaling algorithm to solve this convex cost flow problem using Lemmas 4 and 5. We proved that our implementation on the convex cost flow problem, in practice, performs fewer operations than our implementation on the parallel network.

Acknowledgments

The author is thankful to the editor and two anonymous reviewers for their constructive comments, which helped to improve the manuscript.

References

1. Li, B., Springer, J., Bebis, G. and Gunes, M.H. "A survey of network flow applications", *Journal of Network and Computer Applications*, **36**, pp. 567-581 (2013).
2. Barlet, P. and Cabellos, A. "Analysis of the impact of sampling on NetFlow traffic classification", *Methodology*, **55**, pp. 1083-99 (2010).
3. Hoque, N., Bhuyan, M.H., Baishya, R.C., Bhattacharyya, D.K. and Kalita, J.K. "Network attacks: Taxonomy, tools and systems", *Journal of Network and Computer Applications*, **40**, pp. 307-324 (2014).
4. Lee, M., Hajjat, M., Kompella, R.R. and Rao, S. "Preserving application structure in sampled flow measurements", In *INFOCOM, 2011 Proceedings IEEE*, pp. 2354-62 (April, 2011).
5. Lee, Y., Kang, W. and Lee, Y. "A hadoop-based packet trace processing tool", In *Proceedings of the Third International Conference on Traffic Monitoring and Analysis*, TMA11. Berlin, Heidelberg: Springer-Verlag, pp. 51-63 (2011).
6. Li, Y. "Study of the monitoring model for securities trading network Quality of service", In *2nd International Conference on Information Science and Engineering (ICISE)*, pp. 1-4 (2010).
7. Liang, C. and Jian, G. "Fast application-level traffic classification using net flow records", *Journal on Communications*, **33**, pp. 145-52 (2012).
8. Nie, L., Jiang, D. and Guo, L. "A convex optimization-based traffic matrix estimation approach in IP-over-WDM backbone networks", *Journal of Network and Computer Applications*, **50**, pp. 32-38 (2015).
9. Shelley, D.S. and Gunes, M.H. "Gerbilsphere: inner sphere network visualization", *Computer Networks*, **56**, pp. 1016-28 (2012).
10. Zhang, X. and Ding, W. "Flow-aggregation accelerating strategy for TCP traffic", *Journal of Networking*, **9**, pp. 1416-1425 (2014).
11. Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ (1993).
12. Orlin, J.B. "Max flows in O(mn) time, or better", *STOC*, pp. 765-774 (2013).
13. Furkerson, D.R. "A network flow feasibility theorem and combinatorial applications", *Canadian Journal of Mathematics*, **11**, pp. 440-451 (1959).
14. Hassin, R. "The minimum cost flow problem: A unifying approach to dual algorithms and a new tree-search algorithm", *Mathematical Programming*, **25**, pp. 228-239 (1983).
15. Hoffman, A.J. "Some recent applications of the theory of linear inequalities to extremal combinatorial analysis", In *American Mathematical Society*, R. Bellman and M. Hall, Eds., pp. 113-127 (1960).
16. McCormick, S.T. "How to compute least infeasible flows", *Mathematical Programming*, **78**, pp. 179-194 (1997).
17. Ervolina, T.R. and McCormick, S.T. "Two strongly polynomial cut canceling algorithms for minimum cost network flow", *Discrete Applied Mathematics*, **46**, pp. 133-165 (1993).
18. Radzik, T. and Goldberg, A.V. "Tight bounds on the number of minimum-mean cycle cancelation and related results", *Algoritmica*, **11**, pp. 226-242 (1994).

19. Ghiyasvand, M. "An $O(mn\log(nU))$ time algorithm to solve the feasibility problem", *Applied Mathematical Modelling*, **35**, pp. 5276-5285 (2011).

20. Main, R.A. "Infeasibility analysis using Claudia-I", Technical Report, BP Oil International (1993).

21. Main, R.A. "Infeasibility analysis using Claudia-II", Technical Report, BP Oil International (1993).

22. Salehi Fathabadi, H. and Ghiyasvand, M. "A new algorithm for solving the feasibility problem of a network flow", *Applied Mathematics and Computation*, **192**(2), pp. 429-438 (2007).

23. Greenberg, H.J. "Diagnosing infeasibility for min-cost network flow models, Part I: Dual infeasibility", *IMA Journal of Mathematics in Management*, **1**, pp. 99-109 (1987).

24. Greenberg, H.J. "Diagnosing infeasibility for min-cost network flow models, Part II: primal infeasibility", *IMA Journal of Mathematics Applied in Business and Industry*, **2**, pp. 1-12 (1988).

25. Greenberg, H.J., *A Computer-Assisted Analysis System for Mathematical Programing Models and Solutions: A User's Guide for Analyze*, Kluwer Academic Publications, Boston (1993).

26. Aggarwal, C., Ahuja, R.K., Hao, J. and Orlin, J.B. "Diagnosing infeasibilities in network flow problems", *Mathematical Programing*, **81**, pp. 263-280 (1992).

27. Ghiyasvand, M. "A new approach for computing a most positive cut using the minimum flow algorithms", *Applied Mathematics and Computation*, **176**(1), pp. 27-36 (2006).

28. Chinneck, J.W. "Finding the most useful subset of constraints for analysis in an infeasible linear program", Technical Report SCE-93-07, Systems and Computer Engineering, Carleton University, Ottawa, Canada (1993).

29. Chinneck, J.W. "MINOS(IIS): Infeasibility analysis using minos", *Computers and Operations Research*, **21**, pp. 1-9 (1994).

30. Chinneck, J.W. "Localizing and diagnosing infeasibilities in networks", *INFORMS Journal on Computing* (1996).

31. Chinneck, J.W. and Dravnieks, F.W. "Locating minimal infeasible constraint sets in linear programs", *ORSA Journal on Computing*, **3**, pp. 157-168 (1991).

32. Jiang, D., Xu, Z. and Xu, H. "A novel hybrid prediction algorithm to network traffic", *Annals of Telecommunications*, **70**(9), pp. 427-439 (2015).

33. Jiang, D., Xu, Z. and Xu, H. "Multi-scale anomaly detection for high-speed network traffic", *Transactions on Emerging Telecommunications Technologies*, **26**(3), pp. 308-317 (2015).

34. Jiang, D., Xu, Z., Liu, J. and Zhao, W. "An optimization-based robust routing algorithm to energy-efficient networks for cloud computing", Accepted by *Telecommunication Systems*, **63**, pp. 89-98 (2016).

35. Jiang, D., Zhao, Z., Xu, Z., Yao, C. and Xu, H. "How to reconstruct end-to-end traffic based on time-frequency analysis and artificial neural network", *AEU-International Journal of Electronics and Communications*, **68**(10), pp. 915-925 (2014).

36. Edmonds, I. and Karp, R.M. "Theoretical improvements in algorithmic efficiency for network flow problems", *Journal of the Association on Computing Machinery*, **19**, pp. 248-264 (1972).

37. Goldberg, A.V. and Tarjan, R.E. "Finding minimum-cost circulations by successive approximation", *Math. Oper. Res*, **16**, pp. 430-466 (1990).

38. Orlin, J.B. "A faster strongly polynomial minimum cost flow algorithm", *Oper. Res*, **41**, pp. 338-350 (1993)

39. Vygen, J. "On dual minimum cost flow algorithms", *Mathematical Methods of Operations Research*, **56**, pp. 101-126 (2002).

40. Ahuja, R.K., Goldberg, A.V., Orlin, J.B. and Tarjan, R.E. "Finding minimum-cost flows by double scaling", *Mathematical Programming*, **53**, pp. 243-266 (1992).

41. Frangioni, A. and Manca, A. "A computational study of cost reoptimization for min-cost flow problems", *INFORMS J. Comput*, **18**, pp. 61-70 (2006).

42. Ervolina, T.R. and McCormick, S.T. "Two strongly polynomial cut canceling algorithms for minimum cost network flow", *Dicreate Applied Mathematics*, **46**, pp. 133-165 (1993).

43. Georgiadis, L., Goldberg, A.V., Tarjan, R.E. and Werneck, R.F. "An experimental study of minimum mean cycle algorithms", *Proc. 6th International Workshop on Algorithm Engineering and Experiments, SIAM*, pp. 1-13 (2009).

44. Ghiyasvand, M. "A Geometrical explanation to the optimality concept of minimum cost flows", *Scientia Iranica*, **23**, pp. 3063-3071 (2016).

45. Geranis, G., Paparrizos, K. and Sifaleras, A. "On a dual network exterior point simplex type algorithm and its computational behavior", *RAIRO-Operations Research*, **46**(3), pp. 211-234 (2012).

46. Ghiyasvand, M. "A polynomial-time implementation of Pla's method to solve the MCT problem", *Advances in Computational Mathematics and Its Applications*, **1**(2), pp. 104-109 (2012).

47. Kiraly, Z. and Kovacs, P. "Efficient implementations of minimum-cost flow algorithms", *Acta Universitatis Sapientiae*, **4**, pp. 67-118 (2012).

48. Ghiyasvand, M. "A new polynomial-time implementation of the out-of-kilter algorithm using Minty[s lemma", *Control and Cybernetics*, **43**(1), pp. 79-94 (2014).

49. Kovacs, P. "Minimum-cost flow algorithms: An experimental evaluation", *Optimization Methods and Software*, **30**(1), pp. 94-127 (2015).

50. Ghiyasvand, M. "An $O(m(m + n\log n)\log(nC))$-time algorithm to solve the minimum cost tension problem", *Theoretical Computer Science*, **448**, pp. 47-55 (2012).

51. Sifaleras, A. "Minimum cost network flows: Problems, algorithms, and software", *Yugoslav Journal of Operations Research*, **23**(1), pp. 3-17 (2013).

Biography

Mehdi Ghiyasvand was born in Hamedan, Iran in 1975. He obtained MS and PhD degrees in Operations Research from Tehran University and spent a Post-Doctoral term at MIT, USA. He is now an Associate Professor in Bu-Ali Sina University.