Bulletin of the *Transilvania* University of Braşov • Vol 9(58), No. 1 - 2016
Series III: Mathematics, Informatics, Physics, 97-110

A MODIFIED ALGORITHM FOR SOLVING THE INVERSE MINIMUM COST FLOW PROBLEM UNDER THE BOTTLENECK-TYPE HAMMING DISTANCE

Massoud AMAN[2], Hassan HASSANPOUR[3] and Javad TAYYEBI[*1]

Abstract

Given an instance of the minimum cost flow problem, the corresponding inverse problem is to modify the arc costs as little as possible so that a given feasible flow becomes optimal to the modified minimum cost flow problem. In this article, we study this inverse problem in that the modifications are measured by the weighted bottleneck-type Hamming distance. We present a new efficient algorithm to solve the inverse problem. Finally, we theoretically compare the proposed algorithm with the previous one.

2010 *Mathematics Subject Classification:* $90C27$, $03D15$.
Key words: Inverse problem, Hamming distance, minimum cost flow problem.

1 Introduction

For a particular optimization problem and a given feasible solution $\mathbf{x}^0$, the inverse problem is to adjust some parameters of the optimization problem as little as possible to make $\mathbf{x}^0$ form an optimal solution to the modified problem. The modifications can be measured by different distances as the l_1, l_2 and l_∞ norms and also, the weighted Hamming distances. The concept of inverse problems was first proposed by Tarantola in geophysical sciences [11]. Subsequently, Burton and Toint [5, 6] made use of this concept in combinatorial optimization and considered the inverse shortest path problem. Afterwards, several types of inverse combinatorial optimization problems were considered by many authors (see [7, 9] for a survey).
Two versions of the inverse minimum cost flow problems are studied in literature:

[2]Department of Mathematics, Faculty of Mathematical Sciences and Statistics, *Birjand* University, Birjand, Iran, e-mail: mamann@birjand.ac.ir
[3]Department of Mathematics, Faculty of Mathematical Sciences and Statistics, *Birjand* University, Birjand, Iran, e-mail: hhassanpour@birjand.ac.ir
[1]* *Corresponding author*, Department of Industrial Engineering, Birjand University of Technology, *Birjand*, Iran, e-mail: javadtayyebi@birjandut.ac.ir& javadtayyebi@birjand.ac.ir

the capacity inverse minimum cost flow problem and the (cost) inverse minimum cost flow problem. In the capacity inverse minimum cost flow problem, we adjust the arc capacities so that a feasible solution becomes optimal to the modified problem. This problem is considered in [8, 13]. For the l_1 norm and the sum-type Hamming distance, it is shown that the problem is strongly NP-complete even on bipartite networks. For the l_∞ norm and the bottleneck-type Hamming distance, algorithms are presented to solve the problem in strongly polynomial time. In the (cost) inverse minimum cost flow problem, the initial cost vector is modified such that a given feasible flow is optimal with respect to the new cost vector. Zhang and Liu [15] used the optimality conditions of the linear programming problems and proposed an efficient algorithm for solving the inverse problem under the l_1 norm. Ahuja and Orlin [2, 3] showed that the inverse of a linear programming problem is a new linear programming problem and considered the inverse minimum cost flow problem, a special case of inverse linear programming problems. When the modifications of the cost vector are measured by the l_1 norm, they showed the inverse problem reduces to a unit capacity minimum cost flow problem. For the l_∞ norm, they converted the inverse problem into a minimum cost-to-time ratio cycle problem. Jiang et al. [10] considered the inverse minimum cost flow problem under both the sum-type and the bottleneck-type Hamming distances. For the sum-type case, it is shown that the problem is APX-hard by reduction from the weighted feedback arc set problem. In the bottleneck-type case, they proposed an algorithm for solving the problem. Recently, we showed that their proposed algorithm does not solve correctly the inverse problem due to some incorrect results [12]. Then, we presented two algorithms to solve the inverse problem in $O(m^2n)$ and $O(mn \log n)$ times [12, 14].

In this paper, we propose an algorithm for the bottleneck-type case. This algorithm solves a shortest path problem in each iteration similar to the one presented in [12]. It uses the concepts of sensitivity analysis to find shortest paths by using the shortest paths obtained in the previous iteration efficiently. Its worst-case complexity is $O(m^2n)$ similar to that of the previous one but it runs faster than the previous one in practice because the computational cost of solving shortest path problems decreases.

The rest of this article is organized as follows. In Section 2, we state some preliminaries and notions. In Section 3, we present our proposed algorithm and compare it with the previous one presented in [12]. Finally, we conclude in Section 4.

2 Some fundamental notions and problem definition

In this section, we review some fundamental notions used throughout this article and also, formulate the inverse minimum cost flow problem.

Let $G(V, A, \mathbf{c}, \mathbf{p})$ be a connected and directed network where $V = \{1, 2, \ldots, n\}$ is the set of nodes, A is the set of m arcs, $\mathbf{p}$ is the capacity vector for arcs and $\mathbf{c}$ is the cost vector for arcs. Each node $i \in V$ has an associated supply or demand of value $b(i)$. The well-known minimum cost flow problem minimizes the cost of sending

a flow $\mathbf{x}$ to the network $G(V, A, \mathbf{c}, \mathbf{p})$ under some balancing constraints over the nodes and capacity constraints over the arcs. This problem can be formally stated as follows [1]:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij}x_{ij}, \\ s.t. \quad & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = b_i \quad \forall i \in V, \\ & 0 \le x_{ij} \le p_{ij} \quad \forall (i,j) \in A. \end{aligned} \tag{1}$$

For a network $G(V, A, \mathbf{p}, \mathbf{c})$ and a feasible flow $\mathbf{x^0}$ of the network, the corresponding residual network, denoted by $G'(V, A', \mathbf{x}^0, \mathbf{p}', \mathbf{c}')$, can be constructed by the following algorithm.

Algorithm 1. *([10])*

Step 1 *The node set is still* V.

Step 2 *If* $(i,j) \in A$ *and* $x^0_{ij} < p_{ij}$, *then* $(i,j) \in A'$, $c'_{ij} = c_{ij}$ *and* $p'_{ij} = p_{ij} - x^0_{ij}$.

Step 3 *If* $(i,j) \in A$ *and* $x^0_{ij} > 0$ *then* $(j,i) \in A'$, $c'_{ji} = -c_{ij}$ *and* $p'_{ji} = x^0_{ij}$.

We denote the arc sets created by steps 2 and 3 as A'_1 and A'_2, respectively. Let us introduce some notations for a given feasible flow $\mathbf{x}^0$ of problem (1) as follows:

$$\begin{aligned} L &= \{(i,j) \in A : x^0_{ij} = 0\}; \\ M &= \{(i,j) \in A : 0 < x^0_{ij} < p_{ij}\}; \\ U &= \{(i,j) \in A : x^0_{ij} = p_{ij}\}. \end{aligned}$$

The relationships between the arc sets defined are given in the following.

Property 1.

- $(i,j) \in A'_1$ *if and only if* $(i,j) \in L \cup M$.
- $(i,j) \in A'_2$ *if and only if* $(j,i) \in U \cup M$.

For each arc $(i,j) \in A$, the associated reduced cost is defined as $c^\pi_{ij} = c_{ij} - \pi_i + \pi_j$ where π_i, $i \in V$, is the ith variable of the corresponding dual problem. The following lemma gives the complementary slackness optimality conditions for a given feasible solution [1].

Theorem 1. *A feasible flow* $\mathbf{x}^0$ *is optimal to problem (1) if and only if the following conditions hold:*

- $c^\pi_{ij} \ge 0$ *for each* $(i,j) \in L$;

- $c_{ij}^{\pi} = 0$ *for each* $(i,j) \in M$*;*
- $c_{ij}^{\pi} \leq 0$ *for each* $(i,j) \in U$*;*

for some values of the dual variables π*.*

A special type of problem (1) is the shortest path problem when the upper bounds are infinite, $b(s) = n-1$ and $b(i) = -1, i \in V \backslash \{s\}$, where $s \in V$ is a distinguished node called the source. Each basic feasible solution to the shortest path problem corresponds to a spanning out-tree T rooted at node s. For a given out-tree T and each $j \in V$, $pred(j)$ is a node i so that $(i,j) \in T$. A node j is called a successor of node i if $pred(j) = i$. The descendants of a node i are the node i itself, its successors, successors of its successors, and so on. We denote the descendants of a node i by $des_T(i)$. For a given out-tree T and every nontree arc $(i,j) \in A$, we denote by C_{ij} the unique cycle in $T \cup \{(i,j)\}$. It can be verified that the reduced cost of arc (i,j) equals to $\sum_{(l,k)\in \bar{C}_{ij}} c_{lk} - \sum_{(l,k)\in \underline{C}_{ij}} c_{lk}$ where $\bar{C}_{ij}$ and $\underline{C}_{ij}$ contain respectively arcs of C_{ij} in the direction and the opposite direction of (i,j) [1].
We now define the inverse minimum cost flow problem. For an instance of the minimum cost flow problem and a given feasible flow $\mathbf{x}^0$, the inverse minimum cost flow problem under the bottleneck-type Hamming distance is to replace the cost vector $\mathbf{c}$ with $\mathbf{d}$ so that the following conditions are satisfied [10, 12]:

- $\mathbf{x}^0$ is an optimal flow to the minimum cost flow problem defined on $G(V, A, \mathbf{d}, \mathbf{p})$.
- $-l_{ij} \leq d_{ij} - c_{ij} \leq u_{ij}$ where $l_{ij} \geq 0$ and $u_{ij} \geq 0$ are respectively given bounds for decreasing and increasing c_{ij}. These constraints are called the bound constraints.
- The value $\max_{(i,j)\in A} w_{ij} H(c_{ij}, d_{ij})$ is minimized where $w_{ij} > 0$ is a penalty for modifying c_{ij} and $H(c_{ij}, d_{ij})$ is the Hamming distance between c_{ij} and d_{ij} , i.e., $H(c_{ij}, d_{ij}) = 1$ if $c_{ij} \neq d_{ij}$ and $H(c_{ij}, d_{ij}) = 0$ otherwise.

By using Theorem 1, we can formulate the inverse minimum cost flow problem as follows :

$$\begin{aligned} &\min \max_{(i,j)\in A} w_{ij} H(c_{ij}, d_{ij}), \\ &d_{ij}^{\pi} \geq 0 \quad \forall (i,j) \in L, \\ &d_{ij}^{\pi} = 0 \quad \forall (i,j) \in M, \\ &d_{ij}^{\pi} \leq 0 \quad \forall (i,j) \in U, \\ &-l_{ij} \leq d_{ij} - c_{ij} \leq u_{ij}, \end{aligned} \tag{2}$$

where $d_{ij}^{\pi} = d_{ij} - \pi_i + \pi_j$ for some values of the dual variables π_i. In problem (2), d_{ij}'s and π_i's are variables to be determined. Without any loss of generality, we henceforth assume that the values w_{ij}'s are distinct. This assumption guarantees that each w_{ij} is only associated with one arc (i,j).

3 Algorithm

In this section, we consider problem (2) and propose an efficient algorithm. Sort the objective values of problem (2) in the nondecreasing order: let $w_0 = 0 \leq w_1 \leq w_2 \leq \ldots \leq w_m$ denote the sorted list of the objective values of the problem. Each element of the sorted list (except $w_0 = 0$) equals to the penalty of an arc. If the optimal objective value is w_0, then $\mathbf{x}^0$ is optimal to the initial network; else, the initial network has to be modified.
The proposed algorithm is to find the least index $k \in \{0, 1, \ldots, m\}$ so that there exists a feasible solution of problem (2) with the objective value w_k. Obviously, a feasible solution with this property is optimal to problem (2). Suppose that $\mathbf{d}$ is such a feasible solution. Since the objective value of $\mathbf{d}$ is equal to w_k, it follows that $d_{ij} = c_{ij}$ for each $(i, j) \in A$ with $w_{ij} > w_k$. But the component d_{ij} can be any value in the interval $[c_{ij} - l_{ij}, c_{ij} + u_{ij}]$ if $w_{ij} \leq w_k$. We only restrict our attention to a special form of feasible solutions and look for such an optimal solution. Let us provide some more details concerning this special form by considering the following cases:

Case 1 ($(i, j) \in L$ **with** $w_{ij} \leq w_k$) In this case, d_{ij} must satisfy the inequality $d^{\pi}_{ij} = d_{ij} - \pi_i + \pi_j \geq 0$ if problem is feasible. Hence, we set d_{ij} equal to its upper bound $c_{ij} + u_{ij}$ to obtain the largest value of d^{π}_{ij}.

Case 2 ($(i, j) \in U$ **with** $w_{ij} \leq w_k$) In this case, $d^{\pi}_{ij} = d_{ij} - \pi_i + \pi_j \leq 0$ if problem is feasible. Thus, we set d_{ij} equal to its lower bound $c_{ij} - l_{ij}$ to obtain the minimum value of d^{π}_{ij}.

Case 3 ($(i, j) \in M$ **with** $w_{ij} \leq w_k$) In this case, theorem 1 implies that $d_{ij} = \pi_i - \pi_j$.

Case 4 ($(i, j) \in A$ **with** $w_{ij} > w_k$) We set $d_{ij} = c_{ij}$ to obtain a solution with the objective value less than or equal to w_k.

Now, we are in the position to introduce the special form of feasible solutions. Suppose that $(\mathbf{d}, \pi)$ is a feasible solution to problem (2) with objective value less than or equal to w_k. We define a new cost vector $\mathbf{d}^{(k)}$ as follows:

$$d^{(k)}_{ij} = \begin{cases} c_{ij} & A \backslash A^{(k)}, \\ c_{ij} + u_{ij} & (i, j) \in L \cap A^{(k)}, \\ c_{ij} - l_{ij} & (i, j) \in U \cap A^{(k)}, \\ d_{ij} & (i, j) \in M \cap A^{(k)}, \end{cases} \tag{3}$$

where $A^{(k)} = \{(i, j) \in A : w_{ij} \leq w_k\}$. It is obvious that the objective value of $\mathbf{d}^{(k)}$ is at most w_k. The following proposition is on the feasibility of $\mathbf{d}^{(k)}$

Proposition 1. *If the ordered pair $(\mathbf{d}, \pi)$ is a feasible solution to problem (2) with objective value less than or equal to w_k, then $(\mathbf{d}^{(k)}, \pi)$ is also feasible where $\mathbf{d}^{(k)}$ is defined in (3).*

Proof. It is easy to see that $\mathbf{d}^{(k)}$ satisfies the bound constraints by its definition. From the feasibility of $(\mathbf{d}, \pi)$ and the definition of $\mathbf{d}^{(k)}$, the following expressions hold:

- $d_{ij} \leq c_{ij} + u_{ij} = d_{ij}^{(k)} \qquad \forall (i,j) \in L \cap A^{(k)}$;
- $d_{ij} \geq c_{ij} - l_{ij} = d_{ij}^{(k)} \qquad \forall (i,j) \in U \cap A^{(k)}$;
- $d_{ij}^{(k)} = d_{ij} \qquad \forall (i,j) \in (A \backslash A^{(k)}) \cup M$.

These imply that

- $(d_{ij}^{(k)})^{\pi} = d_{ij}^{(k)} - \pi_i + \pi_j \geq d_{ij} - \pi_i + \pi_j = d_{ij}^{\pi} \geq 0 \qquad \forall (i,j) \in L$;
- $(d_{ij}^{(k)})^{\pi} = d_{ij}^{(k)} - \pi_i + \pi_j \leq d_{ij} - \pi_i + \pi_j = d_{ij}^{\pi} \leq 0 \qquad \forall (i,j) \in U$;
- $(d_{ij}^{(k)})^{\pi} = d_{ij}^{(k)} - \pi_i + \pi_j = d_{ij} - \pi_i + \pi_j = d_{ij}^{\pi} = 0 \qquad \forall (i,j) \in M$.

Thus, $(\mathbf{d}^{(k)}, \pi)$ is also feasible. □

Proposition 1 guarantees the feasibility of $(\mathbf{d}^{(k)}, \pi)$ if a feasible solution $(\mathbf{d}, \pi)$ with objective value less than or equal to w_k exists. The feasibility of $\mathbf{d}^{(k)}$ can be identified even without introducing $\mathbf{d}$ and only by finding a convenient vector π. Based on the constraints of problem (2), $(\mathbf{d}^{(k)}, \pi)$ is feasible if the following inequalities system has at least one solution:

$$\begin{aligned}
\pi_i - \pi_j &\leq c_{ij} + u_{ij} && \forall (i,j) \in L \cap A^{(k)}, && \text{(4a)}\\
\pi_i - \pi_j &\leq c_{ij} && \forall (i,j) \in L \backslash A^{(k)}, && \text{(4b)}\\
\pi_i - \pi_j &\geq c_{ij} - l_{ij} && \forall (i,j) \in U \cap A^{(k)}, && \text{(4c)}\\
\pi_i - \pi_j &\geq c_{ij} && \forall (i,j) \in U \backslash A^{(k)}, && \text{(4d)}\\
\pi_i - \pi_j &= d_{ij}^{(k)} && \forall (i,j) \in M \cap A^{(k)}, && \text{(4e)}\\
\pi_i - \pi_j &= c_{ij} && \forall (i,j) \in M \backslash A^{(k)}, && \text{(4f)}\\
d_{ij}^{(k)} &\leq c_{ij} + u_{ij} && \forall (i,j) \in M \cap A^{(k)}, && \text{(4g)}\\
d_{ij}^{(k)} &\geq c_{ij} - l_{ij} && \forall (i,j) \in M \cap A^{(k)}, && \text{(4h)}
\end{aligned}$$

where π_i, $i \in V$, and $d_{ij}^{(k)}$, $(i,j) \in M \cap A^{(k)}$, are unknowns to be determined. By using (4e), we eliminate the unknowns $d_{ij}^{(k)}$. Consequently, the system is converted

into

$$\begin{aligned}
\pi_i - \pi_j &\le c_{ij} + u_{ij} && \forall (i,j) \in L \cap A^{(k)},\\
\pi_i - \pi_j &\le c_{ij} && \forall (i,j) \in L \backslash A^{(k)},\\
\pi_i - \pi_j &\ge c_{ij} - l_{ij} && \forall (i,j) \in U \cap A^{(k)},\\
\pi_i - \pi_j &\ge c_{ij} && \forall (i,j) \in U \backslash A^{(k)},\\
\pi_i - \pi_j &= c_{ij} && \forall (i,j) \in M \backslash A^{(k)},\\
\pi_i - \pi_j &\le c_{ij} + u_{ij} && \forall (i,j) \in M \cap A^{(k)},\\
\pi_i - \pi_j &\ge c_{ij} - l_{ij} && \forall (i,j) \in M \cap A^{(k)},
\end{aligned}$$

or equivalently,

$$\begin{aligned}
\pi_i - \pi_j &\le c_{ij} + u_{ij} && \forall (i,j) \in L \cap A^{(k)},\\
\pi_i - \pi_j &\le c_{ij} && \forall (i,j) \in L \backslash A^{(k)},\\
\pi_j - \pi_i &\le -c_{ij} + l_{ij} && \forall (i,j) \in U \cap A^{(k)},\\
\pi_j - \pi_i &\le -c_{ij} && \forall (i,j) \in U \backslash A^{(k)},\\
\pi_i - \pi_j &\le c_{ij} && \forall (i,j) \in M \backslash A^{(k)},\\
\pi_j - \pi_i &\le -c_{ij} && \forall (i,j) \in M \backslash A^{(k)},\\
\pi_i - \pi_j &\le c_{ij} + u_{ij} && \forall (i,j) \in M \cap A^{(k)},\\
\pi_j - \pi_i &\le -c_{ij} + l_{ij} && \forall (i,j) \in M \cap A^{(k)}.
\end{aligned} \tag{5}$$

It is remarkable that each constraint of system (5) corresponds to an arc (i,j) in the residual network $G'(V, A', \mathbf{x}^0, \mathbf{u}', \mathbf{c}')$ and vice versa. By using Property 1, the inequalities system can be rewritten in the compact form

$$\pi_i - \pi_j \le \bar{c}_{ij}^{(k)} \quad \forall (i,j) \in A', \tag{6}$$

where the vector $\bar{\mathbf{c}}^{(k)}$ is defined by

$$\bar{c}_{ij}^{(k)} = \begin{cases} c_{ij} + u_{ij} & (i,j) \in A_1' \cap A^{(k)}, \\ c_{ij} & (i,j) \in A_1' \backslash A^{(k)}, \\ -c_{ji} + l_{ji} & (i,j) \in A_2' \cap A^{(k)}, \\ -c_{ji} & (i,j) \in A_2' \backslash A^{(k)}, \end{cases} \quad \forall (i,j) \in A'. \tag{7}$$

System (6) is known as a system of difference constraints. We wish to determine whether the system of difference constraints given by (6) has a feasible solution, and if so, we want to identify a feasible solution. Define the network $G(V'', A'', \mathbf{c}^{(k)})$, called the constraint network, as follows:

$$\begin{aligned}
V'' &= V \cup \{s\}\\
A'' &= A' \cup \{(s,j) : j \in V\}\\
c_{ij}^{(k)} &= \begin{cases} \bar{c}_{ij}^{(k)} & (i,j) \in A', \\ 0 & (i,j) \in A'' \backslash A'. \end{cases}
\end{aligned} \tag{8}$$

The following lemma gives us the desired result.

Lemma 1. *([1]) System (6) has a solution π if and only if the shortest path problem defined on $G(V'', A'', \mathbf{c}^{(k)})$ contains no negative cycle and has the finite shortest path distance π_i from node s to each node i.*

Theorem 2.

- *If the shortest path problem defined on the constraint network $G(V'', A'', \mathbf{c}^{(k)})$ has a finite optimal solution with the shortest path distances π_i, $i \in V$, then the solution $(\mathbf{d}^{(k)}, \pi)$ is feasible to problem (2) where $\mathbf{d}^{(k)}$ is defined by (3) and (4e).*
- *If the constraint network $G(V'', A'', \mathbf{c}^{(k)})$ contains a negative cycle, then problem (2) contains no feasible solution with the objective value less than or equal to w_k.*

Proof. The proof of the first part is obvious from the above argument. We only prove the second part by contradiction. Suppose that problem (2) has a feasible solution $(\mathbf{d}, \pi)$ with the objective value less than or equal to w_k. By Proposition 1, the solution $(\mathbf{d}^{(k)}, \pi)$ is also feasible where $\mathbf{d}^{(k)}$ is defined by (3). Consequently, system (6) has at least one solution. Based on Lemma 1, the constraint network $G(V'', A'', \mathbf{c}^{(k)})$ contains no negative cycle which is a contradiction. □

Now, we are in a position to state a feasibility condition to problem (2).

Corollary 1. *Problem (2) is infeasible if the network $G(V'', A'', \mathbf{c}^{(m)})$ contains a negative cycle where $\mathbf{c}^{(m)}$ is defined by (7) and (8) for $k = m$.*

Proof. Since the network $G(V'', A'', \mathbf{c}^{(m)})$ contains a negative cycle, based on Theorem 2, problem (2) contains no feasible solution with the objective value less than or equal to w_m. Therefore, the problem is infeasible because w_m is the greatest objective value. □

We now explain our proposed algorithm with more details. The algorithm contains at most $m+1$ iterations corresponding to the objective values $w_0, w_1, \ldots, w_m$. In the kth iteration, the algorithm solves the shortest path problem defined on the network $G(V'', A'', \mathbf{c}^{(m+1-k)})$, $k = 1, 2, \ldots, m+1$, and one of the following two cases occurs:

The shortest path problem has a finite optimal solution. Let π_i, $i \in V$, be the shortest path distance from node 1 to node i. Based on Theorem 2, the solution $(\mathbf{d}^{(m+1-k)}, \pi)$ is feasible to problem (2) with the objective value w_{m+1-k}. In this case, the algorithm begins the next iteration for finding a feasible solution with objective value better than w_{m+1-k}, if one exists.

The network $G(V'', A'', \mathbf{c}^{(m+1-k)})$ contains a negative cycle. Problem (2) contains no feasible solution with objective value less than or equal to w_{m+1-k} (see Theorem 2). Hence, the algorithm terminates and the feasible solution obtained from the previous iteration is optimal to problem (2).

Let us summarize our discussion. The algorithm finds a sequence of feasible solutions $\mathbf{d}^{(m)}, \mathbf{d}^{(m-1)}, \ldots$, until the current constraint network contains a negative cycle. The last feasible solution found by the algorithm is optimal based on Theorem 2. It is notable that if the algorithm identifies a negative cycle in the first iteration, then problem (2) is infeasible (see Corollary 1).
Due to the little difference between the constraints networks corresponding to two consecutive iterations, we use the concepts of sensitivity analysis to solve the shortest path problems efficiently. Let $T^{(m+1-k)}$ denote the shortest path tree and $\pi_i^{(m+1-k)}$, $i \in V$, be the shortest path distance of node s to node i in the network $G(V'', A'', \mathbf{c}^{(m+1-k)})$ for $k = 1, 2, \ldots, m+1$. At the beginning of the $(k+1)$th iteration, we wish to obtain $T^{(m-k)}$ and $\pi^{(m-k)}$ by using $T^{(m+1-k)}$ and $\pi^{(m+1-k)}$ obtained from the previous iteration. Suppose that $(i_k, j_k) \in A$ is an arc with the penalty w_{m+1-k}, Since the objective values are distinct, $A^{m+1-k} \backslash A^{m-k} = \{(i_k, j_k)\}$. The following cases determine the differences between the constraint networks corresponding to the kth and $(k+1)$th iterations, i.e. $G(V'', A'', \mathbf{c}^{(m+1-k)})$ and $G(V'', A'', \mathbf{c}^{(m-k)})$, respectively:

- If $(i_k, j_k) \in L$, then the difference is in $(i_k, j_k) \in A'$ because $c_{i_k j_k}^{(m+1-k)} = c_{i_k j_k} + u_{i_k j_k}$ and $c_{i_k j_k}^{(m-k)} = c_{i_k j_k}$.

- If $(i_k, j_k) \in U$, then the difference is in $(j_k, i_k) \in A'$ because $c_{j_k i_k}^{(m+1-k)} = -c_{i_k j_k} + l_{i_k j_k}$ and $c_{j_k i_k}^{(m-k)} = -c_{i_k j_k}$.

- If $(i_k, j_k) \in M$, then the difference is in two arcs $(i_k, j_k), (j_k, i_k) \in A'$ because $c_{i_k j_k}^{(m+1-k)} = c_{i_k j_k} + u_{i_k j_k}$ and $c_{j_k i_k}^{(m+1-k)} = -c_{i_k jk} + l_{i_k j_k}$ but $c_{i_k j_k}^{(m-k)} = c_{i_k j_k}$ and $c_{j_k i_k}^{(m-k)} = -c_{i_k j_k}$.

Therefore, the difference between $G(V'', A'', \mathbf{c}^{(m+1-k)})$ and $G(V'', A'', \mathbf{c}^{(m-k)})$ is at most in two arcs. Note that the modified costs of (i_k, j_k) and (j_k, i_k) in $G(V'', A'', \mathbf{c}^{(m-k)})$ are less than or equal to those in $G(V'', A'', \mathbf{c}^{(m+1-k)})$.
Assume that the difference between $G(V'', A'', \mathbf{c}^{(m+1-k)})$ and $G(V'', A'', \mathbf{c}^{(m-k)})$ is the cost of $(i_k, j_k) \in L$. The other cases can be discussed similarly. We now state how to obtain the current shortest path tree $T^{(m-k)}$ by considering the following cases.

$(i_k, j_k) \notin T^{(m+1-k)}$**:** If the current reduced cost of (i_k, j_k), i.e., $(c^{(m-k)})_{i_k j_k}^{\pi} = c_{i_k j_k}^{(m-k)} - \pi_{i_k}^{(m+1-k)} + \pi_{j_k}^{(m+1-k)}$, is nonnegative, then the optimality conditions hold and the network modification does not change the shortest path tree. thus, $T^{(m-k)} = T^{(m+1-k)}$ and $\pi^{(m-k)} = \pi^{(m+1-k)}$. Else, one of the following two cases occur: (I) The network $G(V'', A'', \mathbf{c}^{(m-k)})$ contains the negative cycle $C_{i_k j_k}$ when $i_k \in des_T(j_k)$. In this case, the algorithm terminates because problem (2) has no feasible solution with objective value less than or equal to w_{m-k} based on Theorem 2. (II) If $i_k \notin des_T(j_k)$, then the network contains no negative cycle and consequently, $T^{(m-k)}$ and $\pi^{(m-k)}$

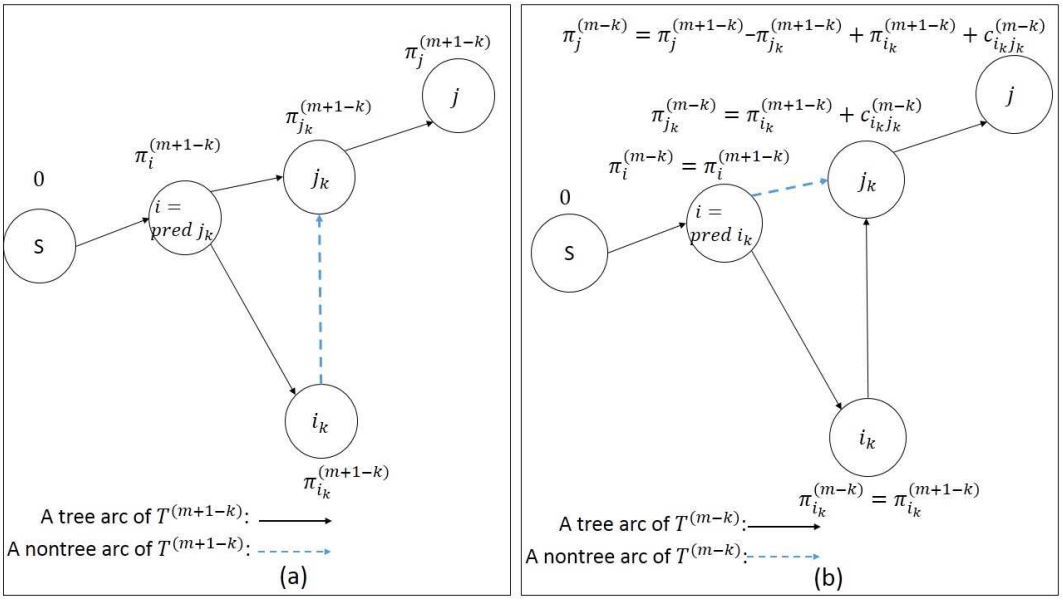

Figure 1: (a) The network $G(V'', A'', \mathbf{c}^{(m+1-k)})$; (b) The network $G(V'', A'', \mathbf{c}^{(m-k)})$.

can be obtained as follows (see Figure 1):

$$T^{(m-k)} = (T^{(m+1-k)} \backslash \{(pred(j_k), j_k)\}) \cup \{(i_k, j_k)\}$$

and

$$\pi_j^{(m-k)} = \begin{cases} \pi_j^{(m+1-k)} - \pi_{j_k}^{(m+1-k)} + \pi_{i_k}^{(m+1-k)} + c_{i_k j_k}^{(m-k)} & j \in des_{T^{(m+1-k)}}(j_k), \\ \pi_j^{(m+1-k)} & j \notin des_{T^{(m+1-k)}}(j_k), \end{cases}$$

for each $j \in V$.

$(i_k, j_k) \in T^{(m+1-k)}$**:** $T^{(m-k)}$ may be quite different from $T^{(m+1-k)}$ if the cost of (i_k, j_k) is modified. Thus, we solve the shortest path problem defined on $G(V'', A'', \mathbf{c}^{(m-k)})$ without using $T^{(m+1-k)}$. One can apply the FIFO label-correcting algorithm as a subroutine to solve the shortest path problem or to identify a negative cycle [1].

Summarily, if the difference between the constraint networks is in a tree arc, then the algorithm solves the shortest path problem and else, it identifies the shortest path tree by using the shortest path tree of the previous iteration.
Now, we are ready to state formally our proposed algorithm for solving problem (2).

Algorithm 2.

Initialization: *Set* $k = 1$.

Step 1: *Solve the shortest path problem defined on* $G(V'', A'', \mathbf{c}^{(m+1-k)})$ *by the FIFO label-correcting algorithm. If the network contains a negative cycle, then stop because problem (2) is infeasible (see Corollary 1). Else, set* T *to the obtained shortest path tree and* π_i *to the shortest path distance from* s *to* i *for each* $i \in V$ *and go to Step 2.*

Step 2: *Determine arc* (i_k, j_k) *with the penalty* w_{m+1-k}. *If* $(i_k, j_k) \in T$ *or* $(j_k, i_k) \in T$, *then go to Step 3 and else, go to Step 4.*

Step 3: *Solve the shortest path problem defined on* $G(V'', A'', \mathbf{c}^{(m-k)})$ *by the FIFO label-correcting algorithm. If the problem has a finite optimal solution, then set* T *to the obtained shortest path tree and* π_i *to the shortest path distance from* s *to* i *for each* $i \in V$ *and go to Step 5. Else, go to Step 6.*

Step 4: *If* $(i_k, j_k) \in L \cup M$ *and the reduced cost of* (i_k, j_k) *is negative, then*

- *if* $i_k \in des_T(j_k)$, *then go to Step 6;*
- *else, update* T *and* π *as follows:* $\pi_j = \pi_j - \pi_{j_k} + \pi_{i_k} + c_{i_k j_k}$ *for each* $j \in des_T(j_k)$ *and* $T = (T \backslash \{pred(j_k), j_k)\}) \cup \{(i_k, j_k)\}$ *and go to Step 5.*

If $(i_k, j_k) \in U \cup M$ *and the reduced cost of* (j_k, i_k) *is negative, then*

- *if* $j_k \in des_T(i_k)$, *then go to Step 6;*
- *else, update* T *and* π *as follows:* $\pi_j = \pi_j - \pi_{i_k} + \pi_{j_k} - c_{i_k j_k}$ *for each* $j \in des_T(i_k)$ *and* $T = (T \backslash \{pred(i_k), i_k)\}) \cup \{(j_k, i_k)\}$ *and go to Step 5.*

Step 5: *If* $k > m$, *then stop because the initial cost vector* $\mathbf{c}$ *with the objective value* $w_0 = 0$ *is optimal to problem (2); else, set* $k = k + 1$ *and go to Step 2.*

Step 6: *Stop and construct* $\mathbf{d}^{(m+1-k)}$ *by using (3) and (4e). The vector* $(\mathbf{d}^{(m+1-k)}, \pi)$ *is an optimal solution to problem (2) with the objective value* w_{m+1-k}.

We now analyze the complexity of Algorithm 2. Since the objective values $w_m, w_{m-1}, \ldots, w_0$ are traversed one by one, the number of iterations is at most $m + 1$. In each iteration, either the FIFO label-correcting algorithm runs or the shortest path tree and the shortest distances are updated. The complexity of the FIFO label-correcting algorithm is $O(mn)$ and the computational cost of updating is at most $O(n)$. Therefore, the bottleneck operation in Algorithm 2 is the number of running the FIFO label-correcting algorithm. So we have established the following result.

Theorem 3. *Algorithm 2 solves problem (2) in* $O(m^2 n)$ *time.*

3.1 Comparison of the two algorithms

In [12], we presented an algorithm for solving problem (2) with the same complexity $O(m^2 n)$. Now, we compare Algorithm 2, called the backward algorithm, with the one presented in [12] which is called the forward algorithm.

The forward algorithm traverses the networks $G(V'', A'', \mathbf{c}^{(k-1)})$ instead of the networks $G(V'', A'', \mathbf{c}^{(m+1-k)})$ for $k = 1, 2, \ldots, m+1$. The backward algorithm finds a feasible solution $\mathbf{d}^{(m+1-k)}$, in the kth iteration and terminates when a negative cycle is identified while the forward algorithm detects the presence of a negative cycle in each iteration and terminates when a feasible solution $\mathbf{d}^{(k-1)}$ is found. The backward algorithm determines whether or not problem (2) is feasible in the first iteration but the other determines it in the last iteration.
A major difference of both the algorithms is that the forward algorithm uses the FIFO label-correcting algorithm as a subroutine in each iteration while the backward algorithm uses it in some iterations. In fact, in each iteration of the backward algorithm, either the FIFO label-correcting algorithm is used or the shortest path tree of the previous iteration is updated. The first occurs with the probability $\frac{n-1}{m}$ and the second with the probability $\frac{m+1-n}{m}$. Hence, the probability of running the FIFO label-correcting algorithm is relatively small, especially for dense networks. Therefore, the backward algorithm is faster than the forward algorithm in practice.
It is notable that the forward algorithm gives only one feasible solution of problem (2) which is also optimal but the backward algorithm can produce a feasible solution in each iteration. Since the computational cost of finding an optimal solution could be large for large networks, one can use some feasible solutions obtained by the backward algorithm as approximations of optimal solutions whenever the available time for solving the problem is short.

4 Conclusions

In this article, we considered the inverse minimum cost flow problem under the weighted bottleneck-type Hamming distance. An efficient algorithm is proposed for solving the problem. The algorithm solves a shortest path problem in each iteration and uses the concepts of sensitivity analysis to improve the running time.

References

[1] Ahuja, R. K., Maganti, T. L. and Orlin, J. B., *Network flows*, Prentice-Hall, Englewood Cliffs, 1993.

[2] Ahuja, R. K. and Orlin, J. B., *Inverse optimization*, Operation Research **49** (2001), 771-783.

[3] Ahuja, R. K. and Orlin, J. B., *Combinatorial algorithms for inverse network flow problems*, Networks **40**, (2002), 181-187.

[4] Bazaraa, M. S. and Jarvis, J. J., *Linear programming and network flows*, Wiley, New York, 1977.

[5] Burton, D. and Toint, Ph. L., *On an instance of the inverse shortest paths problem*, Mathematical Programming 53 (1992) 45-61.

[6] Burton, D. and Toint, Ph. L., *On the use of an inverse shortest paths algorithm for recovering linearly correlated costs*, Mathematical Programming **63** (1994), 1-22

[7] Demange, M. and Monnot, J., *An introduction to inverse combinatorial problems*, In: Paradigms of Combinatorial Optimization (Problems and New approaches, Wiley, London-Hoboken (UK-USA), Vangelis Th. Paschos (2010) 547-586.

[8] Güler, Ç. and Hamacher, H. W., *Capacity inverse minimum cost flow problem*, J. Comb. Optim. **19** (2010), 43-59.

[9] C. Heuberger, Inverse optimization: A survey on problems, methods, and results, Journal of Combinatorial Optimization 8 (2004) 329-361.

[10] Jiang, Y., Liu., Wuc, B. and Yao, E., *Inverse minimum cost flow problems under the weighted Hamming distance*, European Journal of Operational Research **207** (2010), 50-54.

[11] Tarantola, A., *Inverse problem theory: methods for Data Fitting and model parameter estimation*, Elsevier, Amsterdam, recovering linearly correlated costs, Mathematical Programming **63** (1987), 1-22.

[12] Tayyebi, J. and Aman, M., *Note on Inverse minimum cost flow problems under the weighted Hamming distance*, European Journal of Operational Research **234** (2014), 916-920.

[13] Tayyebi, J. and Aman, M., *Capacity inverse minimum cost flow problem under the Hamming distances*, Iranian Journal of Operations Research, In press (2016).

[14] Tayyebi, J. and Aman, M., *On inverse linear programming problems under the bottleneck-type weighted Hamming distance*, Discrete Applied Mathematics, (2015) DOI: 10.1016/j.dam.2015.12.017.

[15] Zhang, J. and Liu, Z., *Calculating some inverse linear programming problems*, Journal of Computational and Applied Mathematics **72** (1996), 261-273.