mathematics

MDPI

Article

Inverse Generalized Maximum Flow Problems

Javad Tayyebi [1,†] and Adrian Deaconu [2,*,†]

[1] Department of Industrial Engineering, Birjand University of Technology, Birjand 9719866981, Iran; javadtayyebi@birjandut.ac.ir

[2] Department of Mathematics and Computer Science, Transilvania University, Brasov 500091, Romania

* Correspondence: a.deaconu@unitbv.ro; Tel.: +40-744378881

† The authors contributed equally to this work.

Received: 23 August 2019; Accepted: 20 September 2019; Published: 25 September 2019

Abstract: A natural extension of maximum flow problems is called the generalized maximum flow problem taking into account the gain and loss factors for arcs. This paper investigates an inverse problem corresponding to this problem. It is to increase arc capacities as less cost as possible in a way that a prescribed flow becomes a maximum flow with respect to the modified capacities. The problem is referred to as the generalized maximum flow problem (IGMF). At first, we present a fast method that determines whether the problem is feasible or not. Then, we develop an algorithm to solve the problem under the max-type distances in $O(mn \cdot \log n)$ time. Furthermore, we prove that the problem is strongly NP-hard under sum-type distances and propose a heuristic algorithm to find a near-optimum solution to these NP-hard problems. The computational experiments show the accuracy and the efficiency of the algorithm.

Keywords: generalized maximum flow; inverse optimization; NP-hardness; Hamming distance

1. Introduction

In a capacitated network, the conventional maximum flow problem is to look for maximum flow which can be sent from a source to a sink under arc capacity constraints. The flow is conserved on arcs and the flow that enters any node (except the source and the sink) equals the flow leaving it.

In the generalized network flow problem, we have a *gain factor* $\gamma(i,j)$ for every arc. This gain factor represents the amount of flow that arrives at node j if we would send one unit of flow from the node i along arc (i,j). More specifically, if we send $f(i,j)$ units from i to j, then $\gamma(i,j)f(i,j)$ units arrive at node j. These gains or losses can refer to evaporation, energy dissipation, breeding, theft, interest rates, blending, or currency exchange. The generalized maximum flow problem can be formulated as a linear programming problem [1]. The augmenting path algorithm and its variants are first algorithms to be proposed for solving the problem [2,3]. A close relationship between this problem and the minimum cost flow problem [4] is stated in Truemper [5]. This fact clarifies that many algorithms of generalized maximum flow problems are similar to those of minimum cost flow problems. Tardos and Wayne [6] developed the first efficient primal algorithm for solving the problem and extend the algorithm for generalized minimum cost flow problem. However, a strongly polynomial-time algorithm that is not based on general linear programming techniques is given for the generalized flow maximization problem in [7], which using a new variant of the scaling technique. Then, another strongly polynomial-time algorithm is developed in [8]. It is faster and simpler than the preceding ones. It works almost exclusively with integral flows, in contrast to all previous algorithms.

For any optimization problem, one can define a corresponding inverse problem. It is how to modify some parameters, such as costs or capacities, in a way that a prescribed solution of the optimization problem becomes optimal with respect to the new parameters. The objective function of inverse problems is to minimize the distance between the initial and new parameters. The changes can

be calculated by Hamming distances or l_p norms. Due to wide range of applications, many researchers have focused on inverse optimization problems in recent years [9–19]. Let us review some papers concerning the inverse maximum flow problem. Yang et al. [20] presented strongly polynomial-time algorithms to solve the inverse maximum flow problem under l_1 norm. In [21–23], efficient algorithms are proposed to solve the inverse maximum flow problem with lower and upper bounds, considering the norms l_1 and l_∞. Inverse maximum flow problems under the bottleneck-type (H_∞) and sum-type (H_1) Hamming distances are also investigated in [24]. Strongly polynomial algorithms are proposed for these problems. The general result is that the inverse maximum flow problem under l_1, l_2, H_1 and H_∞ can be solved in strongly polynomial time.

The reverse problems are another kind of inverse optimization problem. In the reverse maximum flow problem, the goal is to change arc capacities minimally so that the maximum flow value becomes at least a prescribed value v_0. The problem is studied in [25] under weighted l_∞ norm. The authors presented an efficient algorithm based on the discrete-type Newton method to solve the problem.

In this paper, we study the inverse generalized maximum flow problem (denoted IGMF). First we start with an apriori test with a good complexity of $O(mn)$ which decides whether the problem is feasible or not. Then, we develop an efficient algorithm to solve the problem under the max-type distances l_∞ and H_∞. By a reduction of the vertex cover problem, we prove that the problem under the sum-type distances l_1 and H_1 is strongly NP-hard. This result is interesting because the (ordinary) inverse maximum flow problems under the sum-type distances are solved in strongly polynomial time [23,24]. Finally, we present a heuristic algorithm to find pseudo-optimal solutions to the problem.

We recall the definitions of weighted l_1 and weighted l_∞ norms for the n-dimensional vector x:

$$l_1(x) = \sum_{i=1}^{n} w_i|x_i| \tag{1}$$

$$l_\infty(x) = \max_{i=1,\dots,n} w_i|x_i| \tag{2}$$

where $w_i \geq 0$ is the per unit cost attached to the i-th component. A natural use of the norms is that they can be applied to measure the distance between two vectors x and y as $l_k(x-y)$, $k = 1, \infty$. For this reason, the word "distance" is also used instead of "norm" for these functions.

The definitions of the sum-type and bottleneck-type Hamming distances for two n-dimensional vectors x and y are given as follows:

$$H_1(x,y) = \sum_{i=1}^{n} w_i H(x_i, y_i) \tag{3}$$

$$H_\infty(x,y) = max_{i=1,\dots,n} w_i H(x_i, y_i) \tag{4}$$

where $w_i \geq 0$ is the cost of modification associated with the i-th component. The function H measures the Hamming distance between the real values x_i and y_i which is defined as follows:

$$H(x_i, y_i) = \begin{cases} 0, & \texttt{if } x_i = y_i, \\ 1, & \texttt{if } x_i \neq y_i. \end{cases} \tag{5}$$

The rest of the paper is organized as follows: In Section 2, we focus on the generalized maximum flow problem and state its optimality conditions. In Section 3, we introduce the inverse generalized maximum flow problem and we study its feasibility. In Section 4, we present our proposed algorithm to solve the IGMF for the max-type distances. In Section 5, study IGMF under the sum-type distances l_1 and H_1. We prove that these problems are strongly NP-hard. In Section 6, we present a heuristic algorithm for these problems. In Section 7, we perform several computational experiments to consider the accuracy and efficiency of the heuristic algorithm. Finally, some concluding remarks are given in Section 8.

2. The Generalized Maximum Flow

We denote by $G = (N, A, s, t, u, \gamma)$ a generalized network, where N is a set of n-nodes, A is a set of m directed arcs, s and t are special nodes called the source and the sink, respectively. $u : A \to \mathbb{R}_{\geq 0}$ is the capacity function and $\gamma : A \to \mathbb{R}_{\geq 0}$ is the gain function.

The gain of a path P is denoted by $\gamma(P) = \prod_{a \in P} \gamma(a)$. In the same manner, we define the gain of a cycle. A flow-generating cycle is a cycle C whose gain is more than one, i.e., $\gamma(C) > 1$.

We assume that G has no parallel arcs. Without loss of generality we also assume that the network is symmetric, which means that for each arc $(i, j) \in A$ there is an arc $(j, i) \in A$ possibly with zero capacity. The gain function is antisymmetric, i.e., $\gamma(j, i) = \frac{1}{\gamma(i,j)}$.

A function $f : A \to R$ that satisfies the capacity constraints $f(i, j) \leq u(i, j)$ for every $(i, j) \in A$ and the antisymmetry constraints $f(i, j) = -\gamma(j, i) f(j, i)$ for every $(i, j) \in A$ is called a generalized pseudoflow. The residual excess of a node i, except s, is $e_f(i) = -\sum_{(i,j) \in A} f(i, j)$ (the negative of the flow that leaves node i). If $e_f(i)$ is positive (negative), we say that f has residual excess (deficit) at node i. A pseudoflow f is a flow if it has no residual deficits and residual excesses, except in s and t. For a flow f, we denote its value $v(f) = e_f(t)$ to be the residual excess at the sink.

For a generalized flow f in $G = (N, A, s, t, u, \gamma)$ we can define the residual capacity function

$$g(i, j) = u(i, j) - f(i, j). \tag{6}$$

The residual network is $G_f = (N, A_f, s, t, g, \gamma, e_f)$ in which $A_f = \{(i, j) \in A : g(i, j) > 0\}$. The generalized maximum flow problems in the initial network and the residual network are equivalent together. A path in the residual network from an excess node to the sink is called an augmenting path. A flow-generating cycle together with one path from some nodes of this cycle to the sink is referred to as a generalized augmenting path (GAP). One can increase the flow into the sink by sending flow along GAPs and augmenting paths. We shall take an example now.

Example 1. *In Figure 1a we have a network flow in a generalized network with the source node* 1 *and the sink node* 4*. We suppose that* $e(1) = 8$, $e(2) = e(3) = e(4) = 0$*. It is easy to see that* f *satisfies the capacity constraints and the antisymmetry constraints, so, it is a pseudoflow. Let us calculate the residual excesses:* $e_f(2) = 0 - (-7 + 3 + 4) = 0$, $e_f(3) = 0 - (5 - 9 + 4) = 0$ *and* $e_f(4) = 0 - (-1 - 4) = 5$*. It is clear now that the pseudoflow* f *have not any residual deficits (excesses) and, so, it is a flow in the generalized network from Figure 1a. Obviously,* $v(f) = e_f(4) = 5$*. The corresponding residual network is presented in Figure 1b. In this network, we have a flow-generating cycle:* $1 - 2 - 3 - 1$ *whose gain factor is equal to* $1/2 \times 3 \times 2 = 3 > 1$.

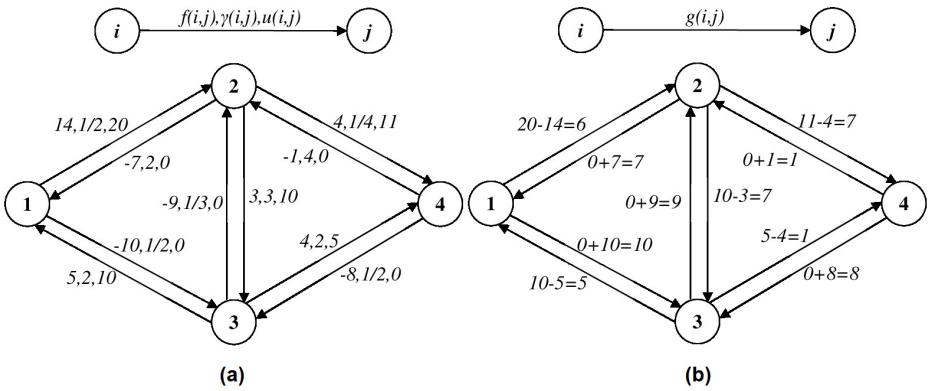

Figure 1. Calculating the residual network for generalized network flows.

The following theorem gives us the optimality conditions for the problem (see [3]):

Theorem 1. *A flow f is optimal in a generalized network G if and only if there is no augmenting path and no GAP in G_f.*

Assume that each arc $(i,j) \in A_f$ is associated with a cost of $c(i,j) = -\log \gamma(i,j)$. To find a GAP in the residual network G_f, we first apply the BFS algorithm to identify the part of the network with nodes which have paths to t. Then, we look after a negative cost cycle C with respect to the arc costs $c(i,j)$ in this part of the network. Notice since

$$\sum_{(i,j)\in C} -\log \gamma(i,j) = -\log\left(\prod_{(i,j)\in C} \gamma(i,j)\right) < 0 \Leftrightarrow \prod_{(i,j)\in C} \gamma(i,j) > 1, \tag{7}$$

it is guaranteed that $C \cup P$ is a GAP where P is a path found by the BFS algorithm from some nodes of C to t. The complexity of this process is $O(mn)$ because the complexity of the BFS algorithm is $O(m)$ and we can use the shortest path algorithm due to Bellman-Ford to find a negative cost in $O(mn)$ [1]. Since the computation of logarithms is time-consuming and inexact in computers, it is not customary to calculate logarithms. However, one can work directly with the gain factors (multiplying gain factors of arcs instead of adding costs of arcs). This yields a modified version of Bellman-Ford algorithm which finds a flow-generating cycle in $O(mn)$ time.

Using the fact that the generalized maximum flow problem is a linear programming, the optimality conditions of linear programming problems can be also used to check the optimality of a flow. For this purpose, suppose that a real number $\pi(i)$ is associated with each node i. Indeed, $\pi(i)$, called the potential of node i, is the dual variable corresponding to the ith balanced constraint. By noting the dual of the problem, it is easy to see that $\pi(s) = 0$ and $\pi(t) = -1$. The potential difference of an arc $(i,j) \in A$ is defined as $d^\pi(i,j) = -\pi(i) + \gamma(i,j)\pi(j)$. The following theorem gives the optimality conditions of a feasible flow to the generalized maximum flow problem.

Theorem 2. *A flow f is optimal to the generalized maximum flow problem if and only if there are node potentials π such that*

- $d^\pi(i,j) \geq 0$ *for every* $(i,j) \in L = \{(i,j) \in A : f(i,j) = 0\}$;
- $d^\pi(i,j) = 0$ *for every* $(i,j) \in F = \{(i,j) \in A : 0 < f(i,j) < u(i,j)\}$;
- $d^\pi(i,j) \leq 0$ *for every* $(i,j) \in U = \{(i,j) \in A : f(i,j) = u(i,j)\}$.

Proof. See Theorem 15.5 in [1]. □

3. Inverse Generalized Maximum Flow Problem

Let $G = (N, A, s, t, u, \gamma)$ be a generalized network. Let f be a feasible flow in the network G. It means that f must satisfy the capacity restrictions, the antisymmetry constraints and it must have no residual deficits and residual excesses (except in s and t).

The inverse generalized maximum flow problem is to change the capacity vector u so that the given feasible flow f becomes a maximum flow in G and the distance between the initial vector of capacities u and the modified vector of capacities, denoted by $\bar{u}$, is minimized:

$$\min dist(u, \bar{u}), \tag{8a}$$

$$f \texttt{ is a maximum flow in } \bar{G} = (N, A, s, t, \bar{u}, \gamma, e), \tag{8b}$$

$$u(i,j) - \delta(i,j) \leq \bar{u}(i,j) \leq u(i,j) + \alpha(i,j),\ \forall (i,j) \in A, \tag{8c}$$

where $\alpha(i,j)$ and $\delta(i,j)$ are the given non-negative numbers to determine bounds on the modifications and $\delta(i,j) \leq u(i,j)$, for each arc $(i,j) \in A$ (see notations of [19]). These values show how much the capacities of the arcs can vary.

It is easy to see that to transform the flow f into a maximum flow in the network $\bar{G}$, it is useless to increase the capacities of the arcs. Therefore, the conditions $\bar{u}(i,j) \leq u(i,j) + \alpha(i,j)$, for each arc $(i,j) \in A$ have no effect and, instead of (8), we consider the following mathematical model:

$$\texttt{min } dist(u,\bar{u}), \tag{9a}$$
$$f \texttt{ is a maximum flow in } \bar{G} = (N,A,s,t,\bar{u},\gamma,e), \tag{9b}$$
$$u(i,j) - \delta(i,j) \leq \bar{u}(i,j),\ \forall (i,j) \in A. \tag{9c}$$

When solving IGMF, if the capacity is changed on arc (i,j), then it is decreased exactly with the amount of $u(i,j) - f(i,j)$ units. If not so, the flow is not stopped from being increased on an augmenting path from s to t or in a GAP that contains the arc (i,j) and the modification of the capacity of (i,j) is useless. This implies that when solving IGMF, it is no need to change the capacities of the arcs from the following set:

$$\widetilde{A} = \{(i,j) \in A | f(i,j) + \delta(i,j) < u(i,j)\}. \tag{10}$$

The set $\widetilde{A}$ is a subset of A_f. Observe that it contains all arcs (i,j) of G with $f(i,j) < 0$, because $\delta(i,j) \leq u(i,j)$.

The above argument together with Theorem 2 suggest a zero-one formulation for IGMF:

$$\texttt{min } dist(u,\bar{u}), \tag{11a}$$
$$-\pi(i) + \gamma(i,j)\pi(j) \leq 0 \qquad \forall (i,j) \in U, \tag{11b}$$
$$-\pi(i) + \gamma(i,j)\pi(j) \leq 0 \qquad \forall (i,j) \in F \backslash \widetilde{A}, \tag{11c}$$
$$-\pi(i) + \gamma(i,j)\pi(j) + My(i,j) \geq 0 \quad \forall (i,j) \in F \backslash \widetilde{A}, \tag{11d}$$
$$-\pi(i) + \gamma(i,j)\pi(j) = 0 \qquad \forall (i,j) \in F \cap \widetilde{A}, \tag{11e}$$
$$-\pi(i) + \gamma(i,j)\pi(j) + My(i,j) \geq 0 \quad \forall (i,j) \in L \backslash \widetilde{A}, \tag{11f}$$
$$-\pi(i) + \gamma(i,j)\pi(j) \geq 0 \qquad \forall (i,j) \in L \cap \widetilde{A}, \tag{11g}$$
$$\pi(t) = -1, \pi(s) = 0, \tag{11h}$$
$$\bar{u}(i,j) = u(i,j) + y(i,j)(f(i,j) - u(i,j)), \tag{11i}$$
$$y(i,j) \in \{0,1\} \qquad \forall (i,j) \in L \cup F, \tag{11j}$$

in which the zero-one variable $y(i,j)$ is defined as $y(i,j) = 1$ if and only if $\bar{u}(i,j) = f(i,j)$. A simple statement of the formulation (11) is that some arcs belonging to $F \backslash \widetilde{A}$ have to be transported to U by setting $\bar{u}(i,j) = f(i,j)$. Consequently, their corresponding constraint, namely $c^{\pi}(i,j) = 0$, is relaxed to $c^{\pi}(i,j) \leq 0$ (see the constraints (11c) and (11e)). Furthermore, setting $\bar{u}(i,j) = 0$, $(i,j) \in L \backslash \widetilde{A}$, removes (i,j) from the network, so the constraint $c^{\pi}(i,j) \geq 0$ is also relaxed (see the constraint (11g)). The formulation (11) is a zero-one linear programming under all the norms l_k and the Hamming distances H_1 and H_∞. So, one can use the zero-one programming technique to solve the problem.

To verify the feasibility of IGMF we construct the network $\widetilde{G} = (N, \widetilde{A}, \gamma)$ in which $\widetilde{A}$ is defined in (10).

Theorem 3. *IGMF is feasible in the generalized network G for a given flow f, if and only if there is no directed path in $\widetilde{G}$ from the node s to the node t and there is no GAP in $\widetilde{G}$.*

Proof. If IGMF is a feasible problem, then it means that there is a vector $\bar{u}$ with $u(i,j) - \delta(i,j) \leq \bar{u}(i,j)$, $f(i,j) \leq \bar{u}(i,j), \forall (i,j) \in A$ and for which the flow f is a maximum flow in the network $\bar{G} = (N,A,s,t,\bar{u},\gamma)$. Since $\widetilde{A} \subseteq \bar{A}_f$, if there exists a directed s-t path in $\widetilde{G}$, it corresponds to a directed path in $\bar{G}_f$, which leads to an augmentation to the flow f in G (contradiction). If there is a GAP in $\widetilde{G}$, then it is a GAP in G_f (contradiction).

Now, for the inverse implication we construct the following capacity vector:

$$u''(i,j) = \begin{cases} u(i,j), & \texttt{if } u(i,j) > f(i,j) + \delta(i,j) \\ f(i,j), & \texttt{otherwise} \end{cases}.$$

It is easy to see that $u(i,j) - \delta(i,j) \leq u''(i,j), f(i,j) \leq u''(i,j), \forall (i,j) \in A$. In the residual network $G''_f = (N, A''_f, r'')$ corresponding to $G'' = (N, A, t, u'', \gamma)$ with respect to the flow f, we have $r''(i,j) = 0$, for every $(i,j) \in (N \times N) \backslash \widetilde{A}$. Hence, $\widetilde{A} = A''_f$. Since there is no directed path from s to t and no GAP in $\widetilde{G}$, it follows that there is no directed path from s to t and no GAP in G''_f. So f is a maximum flow in $G''(N, A, t, u'', \gamma)$. Consequently, u'' is a feasible solution for IGMF. □

The feasibility of IGMF can be determined in $O(mn)$ time complexity because it uses a graph search algorithm in $\widetilde{G}$ which can be done in $O(m)$ time and the adapted Bellman-Ford algorithm in $\widetilde{G}$ which is run in $O(mn)$ time [1].

4. Algorithms for Solving IGMF under Max-Type Distances

Now we study IGMF under max-type distances (denoted IGMFM). This means that in the problem (9) *dist* is defined as follows:

$$dist(u, \bar{u}) = max_{(i,j) \in A} D(u(i,j), \bar{u}(i,j)), \tag{12}$$

where $D : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$. It is easy to see that the bottleneck-type Hamming distance defined in (4) is a particular case of (12) because

$$D(u(i,j), \bar{u}(i,j)) = w(i,j) H(u(i,j), \bar{u}(i,j)),$$

where $w(i,j) \geq 0$ is the cost of modification of the capacity on the arc (i,j).

IGMF under weighted l_∞ norm (denoted IGMF$_\infty$) can be also treated as a particular case of IGMFM. For IGMF$_\infty$, we define

$$D(u(i,j), \bar{u}(i,j)) = w(i,j) |u(i,j) - \bar{u}(i,j)|,$$

where $w(i,j) \geq 0$ is per unit cost of modification of the capacity on the arc $(i,j) \in A$.

Suppose that IGMFM is feasible. The algorithm for IGMFM begins with a set $H = A_f \backslash \widetilde{A}$. So, the elimination of all arcs from H transforms the flow f into a maximum flow in the resulting network. So, we have to find a subset J of H so that if the arcs of J are eliminated then f becomes a maximum flow in the resulting network and the modified capacity vector is optimum for IGMFM. To do this, arcs (i,j) of H are sorted in nondecreasing order by their value $D(u(i,j), \bar{u}(i,j))$. Arcs are eliminated sequentially from H (from arc with the lowest value to the highest) until the arcs of $\widetilde{A} \cup H$ form a graph in which there is no directed path from s to t and there are no GAPs. The arcs that leave the set H are the arcs where the capacities are modified to the value $f(i,j)$. Based on Theorem 1, the flow f is a maximum flow in the resulting network. Let us write the algorithm, formally.

Theorem 4 (the correctness)**.** *The vector u^* found by Algorithm 1 is the optimal solution of IGMFM.*

Proof. Assume that u' is an optimal solution of IGMF with the optimal value $z^* = max_{(i,j) \in A} D(u(i,j), u'(i,j))$. By contradiction, we suppose that

$$max_{(i,j) \in A} D(u(i,j), u^*(i,j)) > z^*. \tag{13}$$

We construct the capacity vector u'' as follows:

$$u''(i,j) = \begin{cases} f(i,j), & \texttt{if } (i,j) \in A \backslash \widetilde{A} \texttt{ and } D(u(i,j), f(i,j)) \leq z^*, \\ u(i,j), & \texttt{otherwise}. \end{cases} \tag{14}$$

It is easy to see that u'' is also optimal solution for IGMF. On the other hand, due to (13), u'' is constructed and tested before constructing u^*. This test failed because the algorithm does not terminate in that iteration. Therefore, u'' is not optimal solution for IGMF which is a contradiction. □

Theorem 5 (the complexity)**.** *The algorithm IGMFM runs in $O(m^2n)$ time.*

Proof. The feasibility initial test is performed in $O(mn)$ time. The vector H can be sorted in $O(m \cdot \log(n))$ time, using an efficient sorting algorithm, for instance QuickSort [26]. The set H contains at most m arcs. If all the arcs of H are eliminated, then f is a maximum flow in the resulting G^*. Therefore, after at most m iterations of the *while* loop f becomes maximum and the algorithm ends. It takes $O(m)$ time to test if there is a directed path from s to t in G^* and $O(mn)$ time to test if there is a GAP in G^*. Consequently, testing if f is a maximum flow takes $O(mn)$ time. Therefore, the total time of the algorithm is $O(m^2n)$. □

Algorithm 1: Algorithm to solve IGMFM

Input: The generalized network $G(N, A, s, t, u, \gamma)$ and flow f.
Output: u^* is the optimal solution of the IGMFM problem.

Construct the residual network $G_f = (N, A_f, s, t, g, \gamma, e_f)$.
Set $\widetilde{u}(i,j) = D(u(i,j), f(i,j)), \quad \forall (i,j) \in A_f$.
Construct the network $\widetilde{G} = (N, \widetilde{A}, \widetilde{u}, \gamma)$.
If The problem is not feasible (there is an s-t directed path or a GAP in $\widetilde{G}$)
    Stop.
End If
Set $u^* = u$ and $H = A_f \backslash \widetilde{A}$.
Sort arcs of H in nondecreasing order with respect to $\widetilde{u}(i,j)$.
While f is not a maximum flow in $G^* = (N, A, s, t, u^*, \gamma)$
    Let (i,j) the first arc from H.
    $H = H \backslash \{(i,j)\}$.
    $u^*(i,j) = f(i,j)$.
End While

We can improve the running time of the IGMFM algorithm by using a "Divide and Conquer" approach. We test the optimality of f after we removed the arcs from the first half of H. We have two situations:

Case 1. If f is a maximum flow in G^*, then we remove only the first quarter of what was initially in H.
Case 2. If f is not a maximum flow in G^*, then we remove also the first half of what remained in H.

The "Divide and Conquer" technique continues until no division can be done any more. The "Divide and Conquer" version of Algorithm 1 is as Algorithm 2.

Since the algorithm deals with the same idea as Algorithm 1, its correctness is obvious. Then, we discuss only about its complexity.

Theorem 6. *The time complexity of the improved IGMFM algorithm is $O(mn \cdot \ log(n))$.*

Proof. The feasibility test can be performed in $O(mn)$ time. The vector H can be sorted in $O(m \cdot log(n))$. Instead of $O(m)$ iterations, the "Divide and Conquer" version has $O(log(m)) = O(log(n))$ iterations. Therefore, the time complexity of "Divide and Conquer" algorithm is $O(mn \cdot log(n))$, since each iteration takes at most $O(mn)$ time. □

Algorithm 2: The "Divide and Conquer" version of Algorithm 1

Input: The generalized network $G(N, A, s, t, u, \gamma)$ and flow f.
Output: u^* is the optimal solution of the IGMFM problem.

Construct the residual network $G_f = (N, A_f, s, t, g, \gamma, e_f)$.
Set $\widetilde{u}(i,j) = D(u(i,j), f(i,j)), \quad \forall (i,j) \in A_f$.
Construct the network $\widetilde{G} = (N, \widetilde{A}, \widetilde{u}, \gamma)$.
If The problem is not feasible (there is an s-t directed path or a GAP in $\widetilde{G}$)
    Stop.
End If
Set $u^* = u$ and $H = A_f \backslash \widetilde{A}$.
Sort the arcs (i,j) of H in nondecreasing order with respect to $\widetilde{u}(i,j)$: let $H = (a^1, a^2, \ldots, a^n)$ is the sorted list.
Set $L = 1$ and $R = n$.
While $L < R$
    Set $M = [\frac{L+R}{2}]$, $u' = u^*$.
    $H' = \{a^{M+1}, a^{M+2}, ..., a^R\}$.
    For $(i', j') \in H \backslash H'$ **do**
        Set $u'(i', j') = f(i', j')$.
    End For
    If f is a maximum flow in $G' = (N, A, s, t, u', \gamma)$
        Set $R = M$.
    Else
        Set $L = M + 1$, $H = H'$ and $u^* = u'$.
    End If
End While

5. IGMF under Sum-Type Distance

In this section, we consider IGMF under the sum-type distances l_1 and H_1. We prove that IGMF under these distances is strongly NP-hard. The proof is based on a reduction from the node cover problem. Let us first recall this problem.

The node cover problem:

Instance, an undirected network $\bar{G}(\bar{N}, \bar{A})$ and a given number k.
Question, is there a set $S \subseteq \bar{N}$ so that $|S| \leq k$ and S is a node cover of $\bar{G}$, i.e., either $i \in S$ or $j \in S$ for every $(i,j) \in \bar{A}$?

Theorem 7. *The inverse generalized maximum flow problem under the l_1 norm is strongly NP-hard.*

Proof. Suppose that an instance of the node cover problem defined on an undirected graph $\bar{G}(\bar{N}, \bar{A})$ is given, where $\bar{N} = \{1, 2, \ldots, n\}$ is the node set and $\bar{A}$ is the arc set. We introduce a bipartite directed network $G(N, A, u, \gamma)$ as follows:

- The network contains two nodes i and i', for each $i \in \bar{N}$. Additionally, we add three nodes s, s', t to the network. Using the notation $\bar{N}' = \{1', 2', \ldots, n'\}$, we have $N = \bar{N} \cup \bar{N}' \cup \{s, s', t\}$.
- For each undirected arc $(i,j) \in \bar{A}$, we add two directed arcs (j', i) and (i', j) to G. We also add (s', i) for $i \in \bar{N}$ and (i', t) for $i' \in \bar{N}'$. We call all these arcs the natural arcs. Then the set of natural arcs is:
$$A_1 = \bigcup_{(i,j) \in \bar{A}} \{(i', j), (j', i)\} \cup \bigcup_{i \in \bar{N}} \{(s', i)\} \cup \bigcup_{i' \in \bar{N}'} \{(i', t)\}.$$
We associate with each $i \in \bar{N} \cup \{s\}$, one arc $(i, i') \in A$. Such arcs are referred to as the artificial arcs, denoted by A_2. Thus, $A = A_1 \cup A_2$. Please note that the underlying undirected graph of G is bipartite.
- The gain of each natural arc is equal to 1 while the gain of each artificial arcs is 2.

- The capacity of each natural arc is equal to $+\infty$. The capacity of each artificial arc is 1.
- $\delta_{ij} = u_{ij}$ for every $(i,j) \in A$.

□

Figure 2 shows an example of how to construct G from G'. Let $f = 0$ be the initial flow. Since the data are polynomially bounded with respect to the problem size, i.e., the similarity assumption is satisfied, we prove the following claim to establish the desired result.

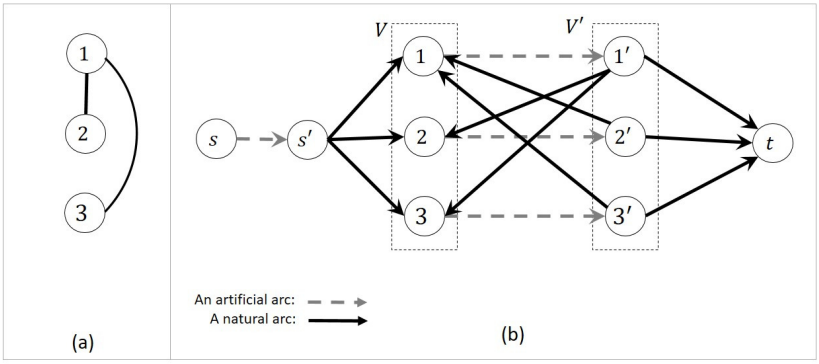

Figure 2. To construct the directed network G from the undirect graph $\bar{G}$.

Claim 1. *The node cover problem in $\bar{G}$ is a yes instance if and only if there is a feasible solution to the inverse generalized maximum flow problem in G with the objective value at most equal to $k+1$.*

Proof. Proof of Claim 1. Suppose that S is a solution to a given yes instance of the node cover problem. We introduce the solution $\bar{u}$ as follows:

$$\begin{aligned}
&\bar{u}(s,s') = 0 (\neq u(s,s') = \frac{1}{2}),\\
&\bar{u}(i,i') = 0 (\neq u(i,i') = 1) \qquad \forall i \in S,\\
&\bar{u}(i,j) = u(i,j) \qquad \forall (i,j) \in A \backslash \{(i,i') : i \in S \cup \{s\}\}.
\end{aligned}$$

It is easy to see that the objective value of $\bar{u}$ is less than or equal to $k + \frac{1}{2}$. Thus, it is sufficient to prove that the residual network with respect to the flow $f = 0$ and the capacity vector $\bar{u}$ contains no st-path and no GAP. Because (s,s') is not in the residual network, we imply that any st-path does not exists in the residual network. Since all gain factors are greater than or equal to 1 and any cycle contains at least an arc (i,i') with $\gamma(i,i') = 2$, it follows that any cycle in the residual network is a part of a GAP. Then, we must prove that the residual network contains no cycle. Any cycle has at least two arcs from $\bar{N}$ to $\bar{N}'$ and at least two arcs from $\bar{N}'$ to $\bar{N}$. Then it contains a path $i - i' - j - j' - k$. Due to this and that $(i,j) \in \bar{A}$, we imply that $i \in S$ or $j \in S$. Equivalently, $\bar{u}(i,i') = 0$ or $\bar{u}(j,j') = 0$. Therefore, the residual network does not contain at least one of two the arcs (i,i') and (j,j'). Then, any cycle cannot exist in the residual network.

Now suppose that $\bar{u}$ is a feasible solution to the inverse generalized maximum flow problem with the objective value $k' < +\infty$. The assumption $k' < +\infty$ guarantees that $\bar{u}(i,j) = u(i,j)$ for each arc (i,j) which has infinity capacity. Hence, only the capacity of artificial arcs can be modified. Consider $S = \{i \in \bar{N} : u(i,i') = 0\}$. We prove that S is a cover of $\bar{G}$ with $|S| \leq k' - 1$. Any st-path in G is $s - s' - i - i' - \ldots - t$. Since the residual network contains no st-path, two cases may occur:

1. (s,s') is not in the residual network.
2. Each arc (i,i') , $i \in \bar{N}$, is not in the residual network.

The first case imposes a cost of 1 on the objective while the second imposes a cost of $n > 1$. Then, due to the optimality of $\bar{u}$, the first case occurred, namely $\bar{u}(s,s') = 0$. On the other hand, we know

that the residual network contains no cycle (GAP). Then, $u(i,i') = 0$ or $u(j,j') = 0$ for each cycle $i - i' - j - j' - i$. This implies that $i \in S$ or $j \in S$ for each $(i,j) \in \bar{A}$. Then, S is a cover of $\bar{G}$ with $|S| = |\{(i,i') : u(i,i') = 0\} \backslash \{(s,s')\}| \leq k' - 1$. This completes the proof. □

A completely similar proof proves that IGMF under H_1 is NP-hard. In the proof. it is sufficient to define the capacity of all arcs equal to 1 and the weight vector w as

$$w(i,j) = \begin{cases} +\infty & (i,j) \in A_1, \\ 1 & (i,j) \in A_2, \end{cases} \qquad (i,j) \in A.$$

Thus, we have the following result.

Theorem 8. *The inverse generalized maximum flow problem under the sum-type Hamming distance is strongly NP-hard.*

6. A Heuristic Algorithm

In this section, we present a heuristic algorithm to obtain pseudo-optimal solutions of IGMF under the sum-type distances.

To remove a GAP C, we must remove an arc $(i,j) \in C$ by setting $\bar{u}(i,j) = f(i,j)$. We use the five following observations to design our algorithm.

1. An arc $(i,j) \in \tilde{A}$ cannot be removed from the residual network because setting $\bar{u}(i,j) = f(i,j)$ violates the bound constraint.
2. A necessary condition for removing arc $(i,j) \in A_f \backslash \tilde{A}$ is that it belongs to at least one GAP.
3. An arc (i,j) belonging to several GAPs has a high priority to be removed because several GAPs are annihilated whenever we remove such an arc.
4. Removing of an arc (i,j) imposes the cost of $u(i,j) - f(i,j)$ ($w(i,j)$) to the objective function under l_1 (H_1).
5. If an arc $(i,j) \in A_f \backslash \tilde{A}$ is on a GAP C so that the other arcs of C belong to $\tilde{A}$, then the arc (i,j) has the greatest priority to be removed because we can eliminate C only by removing (i,j).

We now introduce a preference index $p(i,j)$, $(i,j) \in A$, to determine which one of arcs has high priority to be removed. Based on Observations 3 and 4, an arc is eligible to be removed if

- it is on a greater number of GAPs, and
- it imposes a smaller value of the cost on the objective function.

So, we define

$$p(i,j) = \begin{cases} \frac{v(i,j)}{u(i,j) - f(i,j)} & (i,j) \notin \tilde{A} \text{ and } f(i,j) \neq u(i,j), \\ 0 & (i,j) \in \tilde{A} \text{ or } f(i,j) = u(i,j), \end{cases}$$

for every $(i,j) \in A$ in which $v(i,j)$ is a value to underestimate how many GAPs pass through (i,j). To compute $v(i,j)$'s, we use a successive negative-cycle subroutine. The subroutine initializes $v(i,j) = 0$ for each $(i,j) \in A$. In each iteration, it detects a GAP by using the Reverse Bellman-Ford (RBF) algorithm which has the same process of the standard Bellman-Ford algorithm with this difference which it starts from t and traverses arcs in the opposite direction. The RBF algorithm detects a negative-cycle C' with respect to the arc lengths $c(i,j) = -\log \gamma(i,j)$ in residual network. The output of the RBF algorithm is a negative-cycle C' together with a path P from some nodes of C' to t. Since any negative cycle with respect to $c(i,j)$ is also a generating flow cycle (see (7)), it follows that $C = C' \cup \{P\}$ is a GAP. After detecting a GAP C by the RBF algorithm, the subroutine determines its capacity, i.e., $g_C = \min_{(i,j) \in C} \{g(i,j)\}$. Then, it updates $g(i,j) = g(i,j) - g_C$ and $v(i,j) = v(i,j) + 1$ for each $(i,j) \in C$ and it removes arcs with $g(i,j) = 0$. The process is repeated until any negative cycle is not detected by the RBF algorithm. It is notable that if a GAP C contains no arc of $A_f \backslash \tilde{A}$, then the problem is infeasible

(see Theorem 3). To handle this situation, another output Inf is defined for the subroutine which is a Boolean variable and takes the value of *True* if the subroutine detects this situation. According to Observation 5, another specific situation may occur in which all arcs of the successive negative-cycle GAP belong to $\tilde{A}$, except one arc (i_0, j_0). In this situation, the subroutine sets $v(i_0, j_0) = M$ which is a very big number. Algorithm 3 states the process of the subroutine, formally. Notice that if $v(i, j) = 0$, then there is not any GAP passing through (i, j) (Observation 1). Therefore, the arc (i, j) has the lowest priority to be removed.

The main algorithm in each iteration calls the subroutine for computing $v(i, j)$. Then, it calculates the priority index $p(i, j)$ for each arc $(i, j) \in A$. Finally, it chooses one arc of A with the maximum priority index $p(i, j)$ and remove it from the residual network by setting $\bar{u}(i, j) = f(i, j)$. This process is repeated until the residual network contains no GAP. Our proposed algorithm is given in Algorithm 4.

Algorithm 3: The successive negative-cycle algorithm

Input: The network $G(N, A_f, t, g, c)$, the arc set $\tilde{A}$.
Output: The priority degrees v as well as the Boolean variable Inf which is *True* if the algorithm detects the infeasibility.
For$(i, j) \in A$ **do**
  Set $v(i, j) = 0$.
EndFor
Set $Inf = False$.
While True
  Apply the Reverse Bellman-Ford algorithm starting t to find a negative-cycle C with respect to the arc lengths **c**.
  If There is no negative cycle
    Break.
  End If
  Set $g_C = \min_{(i,j)\in C} g(i, j)$.
  Set $k = 0$.
  For$(i, j) \in C$ **do**
    Set $v(i, j) = v(i, j) + 1$.
    Set $g(i, j) = g(i, j) - g_C$.
    If $g(i, j) = 0$
      Remove (i, j) from A_f.
    End If
    If $(i, j) \notin \tilde{A}$
      Set $k = k + 1$.
      Set $(i_0, j_0) = (i, j)$.
    End If
  End For
  If $k = 0$ %See Theorem 3.
    Set $Inf = True$.
    Break.
  End If
  If $k = 1$ %See Observation 5.
    Set $v(i_0, j_0) = M$ %M is a very big integer
    Break. %(i_0, j_0) is the only arc that can cancel C
  End If
End While

Remark 1. *If we define the priority index as follows:*

$$p(i, j) = \begin{cases} \frac{v(i,j)}{w(i,j)} & (i, j) \notin \tilde{A} \text{ and } f(i, j) \neq u(i, j), \\ 0 & (i, j) \in \tilde{A} \text{ or } f(i, j) = u(i, j), \end{cases}$$

then Algorithm 4 is a heuristic algorithm to obtain a pseudo-optimal solution to the problem under H_1.

Now, let us argument about the complexity of Algorithm 4. The RBF algorithm similar to the standard Bellman-Ford algorithm has a time complexity of $O(mn)$ [1]. Since Algorithms 3 and 4 remove at least one arc in each iteration, then they terminate in $O(m)$ iterations. Hence, we have the following result.

Theorem 9. *Algorithm 4 runs in $O(nm^3)$ time.*

Algorithm 4: A heuristic algorithm to solve IGMF under l_1

Input: The generalized network $G(N, A, s, t, u, \gamma)$ and flow f.
Output: The modified capacity vector $\bar{u}$.
Set $\bar{u} = u$.
Construct the residual network $G_f(V, A_f, s, t, g, \gamma, e_f)$.
Set $\tilde{A} = \emptyset$.
For each $(i, j) \in A_f$ **do**
  If $f(i, j) + \delta(i, j) < u(i, j)$
    Add arc (i, j) to $\tilde{A}$.
  End If
End For
While True
  Apply Algorithm 3. Suppose that the output is (v, Inf).
  If Inf
    Stop because the problem is infeasible.
  End If
  Set $p_{Max} = 0$.
  For $(i, j) \in A$ **do**
    If $g(i, j) > 0$ and $(i, j) \notin \tilde{A}$
      Set $p = \frac{v(i,j)}{u(i,j)-f(i,j)}$.
      If $p_{Max} < p$
        Set $p_{Max} = p$ and $(i_0, j_0) = (i, j)$.
      End If
    End If
  End For
  If $p_{Max} = 0$
    Break.
  Else
    Set $\bar{u}(i_0, j_0) = f(i_0, j_0)$.
    Remove (i_0, j_0) from A_f.
  End If
End While

7. Computational Experiments

In this section, we have conducted a computational study to observe the performance of Algorithm 4. To study its accuracy, we compared the results obtained by Algorithm 4 with the exact optimal solution which is computed by solving the model (11).

The following computational tools were used to develop Algorithm 4 and to solve model (11): Python 2.7.5, Matplotlib 1.3.1, Pulp 1.6.0, and NetworkX 1.8.1. All computational experiments were conducted on a 64-bit Windows 10 with Processor Intel(R) Core(TM) i5 $-$ 3210M CPU @2.50GHz and 4 GB of RAM.

In experiments, we have applied random binomial graphs introduced in [27]. These graphs are determined by two parameters n, the number of nodes, and $p \in [0, 1]$ which is the probability of existing any edge in the graph. In experiments, we have first generated an undirected graph $G(V, A)$ and then have converted it into a directed one by directing any edge (i, j) from i to j if $i < j$. In all experiments, we have assumed that nodes $s = 0$ and $t = n - 1$ are respectively the source and the sink.

All data are generated randomly by using a uniform random distribution as follows:

$$u(i,j) \sim [\mathrm{U}(0,n)] \quad \forall (i,j) \in A,$$
$$\gamma(i,j) \sim \mathrm{U}(0,2) \quad \forall (i,j) \in A.$$

Our experiments show to Algorithm 4 correctly detects the infeasibility of the problem. For this reason, we set $\delta(i,j) = u(i,j)$ for each $(i,j) \in A$ to guarantee the feasibility of instances. To generate a feasible flow, we added a node -1 and an arc $(-1,0)$ with capacity $\frac{n^2}{2}$. Then, we found a maximum flow f from -1 to $n-1$. The flow f saturates $(-1,0)$, then it is a feasible solution with the flow value $\frac{n^2}{2}$. After computing f, we removed the dummy node -1 as well as the arc $(-1,0)$ from the network. The following computational results are the average of running Algorithm 4 on 100 different instances of a class.

We have tested Algorithm 4 on 12 classes of networks which differ from the number of nodes, varying from 5 to 60 (see Table 1). In the tables, z^* is the optimal value and z is the objective value obtain by the algorithm. Since Algorithm 4 finds correctly the solution whenever $z^* = 0$, the results are reported in two cases: total results and results for problems whose optimal value is nonzero. In both the cases, we reported the values $\frac{|z-z*|}{\max\{z,z^*\}} = \frac{z-z*}{z}$ and $\max\{\frac{z}{z^*}, \frac{z^*}{z}\} = \frac{z}{z^*}$. We also tested Algorithm 4 on 9 classes of networks which differ from the density, varying from 0.2 to 1. The results are shown in Table 2.

In another experiment, we run the algorithm 1000 time for $n = 30$ and $p = 0.5$. Then, we classify instances with respect to their optimal value. The results of this experiment are available in Table 3 and are depicted in Figure 3. Observe that the error is lesser when the optimal value is nearer to zero.

Table 1. Average performance statistics of Algorithm 4 for the graphs with $p = \frac{1}{2}$.

n	m	Num. of Prob. with $z^* = 0$	Time (sec)	Total Results		Results for Nonzeros	
				$\frac{z-z^*}{z}$	$\frac{z}{z^*}$	$\frac{z-z^*}{z}$	$\frac{z}{z^*}$
5	4.44	61	0.0013	0.0023	1.0023	0.0059	1.0059
10	21.74	15	0.0059	0.0905	1.0995	0.1064	1.1190
15	52.66	5	0.029	0.187	1.2300	0.1968	1.2450
20	94.24	0	0.0574	0.2214	1.2843	0.2214	1.2843
25	148.6	0	0.1317	0.2605	1.3522	0.2605	1.3522
30	217.26	0	0.3289	0.2955	1.4194	0.2955	1.4194
35	298.31	0	0.4824	0.3002	1.4289	0.3002	1.4289
40	389.11	0	0.6203	0.3132	1.4560	0.3132	1.4560
45	492.93	0	1.2123	0.3133	1.4562	0.3133	1.4562
50	612.66	0	2.4577	0.3266	1.4850	0.3266	1.4850
55	741.76	0	4.238	0.3637	1.5715	0.3637	1.5715
60	884.88	0	5.115	0.3695	1.5860	0.3695	1.5860

Table 2. Average performance statistics of Algorithm 4 for the graphs with $n = 30$.

p	m	Num. of Prob. with $z^* = 0$	Time (sec)	Total Results		Results for Nonzeros	
				$\frac{z-z^*}{z}$	$\frac{z}{z^*}$	$\frac{z-z^*}{z}$	$\frac{z}{z^*}$
0.2	85.7	23	0.0234	0.1248	1.1426	0.1621	1.1935
0.3	129.98	5	0.0564	0.2132	1.271	0.2243	1.2892
0.4	174.21	1	0.1588	0.2876	1.4037	0.2905	1.4094
0.5	217.63	0	0.1944	0.274	1.3774	0.274	1.3774
0.6	261.52	0	0.4493	0.3585	1.5588	0.3585	1.5588
0.7	304.62	0	0.4963	0.2777	1.3845	0.2777	1.3845
0.8	346.68	0	0.6777	0.3606	1.564	0.3606	1.564
0.9	391.04	0	0.8092	0.3201	1.4708	0.3201	1.4708
1	434	0	1.0933	0.3568	1.5547	0.3568	1.5547

Table 3. Average performance statistics of Algorithm 4 for $p = 0.5$ and $n = 30$.

Intervals Containing z^*	**[0, 25)**	**[25, 50)**	**[50, 75)**	**[75, 100)**	**[100, 150)**	**[150, 200)**	**[200, 300)**	**[300, 500)**	**[500, 750)**	**[750, $+\infty$)**
Num. of Obser.	70	217	230	114	141	74	78	50	14	12
$\frac{z-z^*}{z}$	0.047	0.082	0.144	0.237	0.377	0.495	0.601	0.723	0.838	0.888
$\frac{z}{z^*}$	1.049	1.089	1.168	1.311	1.605	1.980	2.506	3.610	6.173	8.929

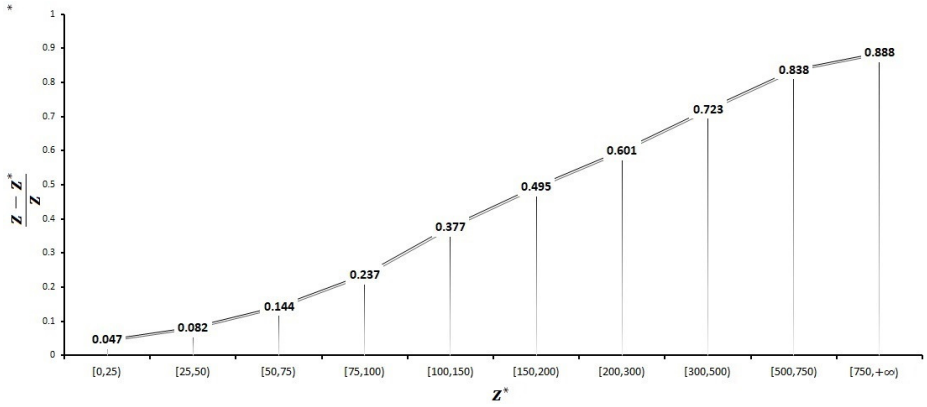

Figure 3. Error chart: X-axis is the intervals containing the optimal value z^* and Y-axis is $\frac{z-z^*}{z}$.

8. Conclusions

In this paper, we have studied two classes of inverse problems: IGMF under max-type distances and under sum-type distances. We have provided a fast initial test of feasibility of IGMF. For the first class we presented polynomial algorithms to solve IGMF in $O(m^2n)$ running time and even in $O(mn \cdot log(n))$ time. We proved that the second class of problems are NP-hard and we presented a heuristic algorithm to solve this kind of problems.

As future works, it will be meaningful that other types of inverse generalized maximum flow problem are investigated. Specifically, one may consider a type of the inverse problem for which gain factors are modified, instead of capacities. This problem can be used to simulate a wide range of real-world applications because gain factor modifications are performed by network restorations.

Author Contributions: All the authors contributed equally to this manuscript.

Funding: This research was funded by Transilvania University of Brasov.

Conflicts of Interest: The Authors declare no conflict of interest.

References

1. Ahuja, R.K.; Magnanti, T.L.; Orlin, J.B. *Network Flows: Theory, Algorithms, and Applications*; Prentice Hall: Englewood Cliffs, NJ, USA, 1993.
2. Jewell, W.S. Optimal flow through networks with gains. *Oper. Res.* **1962**, *10*, 476–499. [CrossRef]
3. Onaga, K. Dynamic programming of optimum flows in lossy communication nets. *IEEE Trans. Circuit Theory* **1966**, *13*, 308–327. [CrossRef]
4. Ciupala, L. A scaling out-of-kilter algorithm for minimum cost flow. *Control Cybern.* **2005**, *34*, 1169–1174.
5. Truemper, K. On max flows with gains and pure min-cost flows. *SIAM J. Appl. Math.* **1977**, *32*, 450–456. [CrossRef]
6. Tardos, E.; Wayne, K.D. Simple Generalized Maximum Flow Algorithms. *Integer. Program. Comb. Optim.* **1998**, *1412*, 310–324.
7. Vegh, L.A. A strongly polynomial algorithm for generalized flow maximization. *Math. Oper. Res.* **2016**, *42*, 179–211. [CrossRef]

8. Olver, N.; Vegh, L.A. A simpler and faster strongly polynomial algorithm for generalized flow maximization. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, ACM 2017, Montreal, QC, Canada, 19–23 June 2017; pp. 100–111.
9. Ahuja, R.K.; Orlin, J.B. Combinatorial Algorithms for Inverse Network Flow Problems. *Networks* **2002**, *40*, 181–187. [CrossRef]
10. Ciurea, E.; Deaconu, A. Inverse Minimum Flow Problem. *J. Appl. Math. Comput.* **2007**, *23*, 193–203. [CrossRef]
11. Demange, M.; Monnot, J. An introduction to inverse combinatorial problems. In *Paradigms of Combinatorial Optimization (Problems and New approaches)*; Paschos, V.T., Ed.; Wiley: London, UK; Hoboken, NJ, USA, 2010.
12. Heuberger, C. Inverse Combinatorial Optimization, A Survey on Problems, Methods, and Results. *J. Comb. Optim.* **2004**, *8*, 329–361. [CrossRef]
13. Karimi, M.; Aman, M.; Dolati, A. Inverse Generalized Minimum Cost Flow Problem Under the Hamming Distances. *J. Oper. Res. Soc. China* **2019**, *7*, 355–364. [CrossRef]
14. Liu, L.; Zhang, J.; Li, C. Inverse minimum flow problem under the weighted sum-type Hamming distance. *Discrete Appl. Math.* **2017**, *229*, 101–112. [CrossRef]
15. Liu, L.; Yao, E. Capacity inverse minimum cost flow problems under the weighted Hamming distance. *Optim. Lett.* **2016**, *10*, 1257–1268. [CrossRef]
16. Nourollahi, S.; Ghate, A. Inverse optimization in minimum cost flow problems on countably infinite networks. *Networks* **2019**, *73*, 292–305. [CrossRef]
17. Tayyebi, J.; Aman, M. On inverse linear programming problems under the bottleneck-type weighted Hamming distance. *Discrete Appl. Math.* **2018**, *240*, 92–101. [CrossRef]
18. Tayyebi, J.; Massoud, A. M. Efficient algorithms for the reverse shortest path problem on trees under the hamming distance. *Yugoslav J. Oper. Res.* **2016**, *27*, 46–60. [CrossRef]
19. Zhang, J., Cai, C. Inverse Problems of Minimum Cuts, ZOR-Math. *Methods Oper. Res.* **1998**, *47*, 51–58. [CrossRef]
20. Yang, C.; Zhang, J.; Ma, Z. Inverse Maximum Flow and Minimum Cut Problems. *Optimization* **1997**, *40*, 147–170. [CrossRef]
21. Ciupala, L.; Deaconu, A. Inverse maximum flow problem in planar networks. *Bull. Transilv. Univ. Brasov Math. Inform. Phys. Ser. III* **2019**, *12*, 113–122.
22. Deaconu, A. The Inverse Maximum Flow Problem Considering L_∞ Norm. *RAIRO-Oper. Res.* **2008**, *42*, 401–414. [CrossRef]
23. Deaconu, A. The inverse maximum flow problem with lower and upper bounds for the flow. *Yugoslav J. Oper. Res.* **2016**, *18*, 13–22. [CrossRef]
24. Liu, L.; Zhang, J. Inverse Maximum Flow Problems under Weighted Hamming Distance. *J. Combin. Optim.* **2006**, *12*, 395–408. [CrossRef]
25. Tayyebi, J.; Mohammadi, A.; Kazemi. S.M.R. Reverse maximum flow problem under the weighted Chebyshev distance. *RAIRO-Oper. Res.* **2018**, *52*, 1107–1121. [CrossRef]
26. Hoare, C.A.R. Algorithm 64: Quicksort. *Commun. ACM* **1961**, *4*, 321. [CrossRef]
27. Erdõs, P.; Rényi, A. On Random Graphs. I. *Publ. Math.* **1959**, *6*, 290–297.