axioms

MDPI

Article

Widest Path in Networks with Gains/Losses

Javad Tayyebi [1], Mihai-Lucian Rîtan [2] and Adrian Marius Deaconu [2,*]

[1] Department of Industrial Engineering, Birjand University of Technology, Industry and Mining Boulevard, Ibn Hesam Square, Birjand 9719866981, Iran; javadtayyebi@birjandut.ac.ir
[2] Depatment of Mathematics and Computer Science, Transilvania University of Braşov, Eroilor St. 29, 500036 Braşov, Romania; mihai.ritan@unitbv.ro
* Correspondence: a.deaconu@unitbv.ro

Abstract: In this paper, the generalized widest path problem (or generalized maximum capacity problem) is studied. This problem is denoted by the GWPP. The classical widest path problem is to find a path from a source (s) to a sink (t) with the highest capacity among all possible s-t paths. The GWPP takes into account the presence of loss/gain factors on arcs as well. The GWPP aims to find an s-t path considering the loss/gain factors while satisfying the capacity constraints. For solving the GWPP, three strongly polynomial time algorithms are presented. Two algorithms only work in the case of losses. The first one is less efficient than the second one on a CPU, but it proves to be more efficient on large networks if it parallelized on GPUs. The third algorithm is able to deal with the more general case of losses/gains on arcs. An example is considered to illustrate how each algorithm works. Experiments on large networks are conducted to compare the efficiency of the algorithms proposed.

Keywords: combinatorial optimization; widest path; maximum capacity path; strongly polynomial algorithms; GPU parallelization

MSC: 68R10; 90B10; 90C35; 68W10

Citation: Tayyebi, J.; Rîtan, M.-L.; Deaconu, A.M. Widest Path in Networks with Gains/Losses. *Axioms* **2024**, *13*, 127. https://doi.org/10.3390/axioms13020127

Academic Editors: Ivan Mauricio Amaya-Contreras and José Carlos Ortiz-Bayliss

Received: 22 January 2024
Revised: 14 February 2024
Accepted: 16 February 2024
Published: 18 February 2024

1. Introduction

Consider a directed and connected network $G = (V,\ A,\ u)$, where V represents the set of nodes, A represents the set of arcs (each arc $a = (i,\ j)$ starts from node i and terminates at node j), and u is a capacity function mapping arcs to non-negative real numbers. Within this network, there are two special nodes: s, referred to as the source node; and t, referred to as the sink node. Let n denote the total number of nodes in the network ($|V|$) and m represent the number of arcs ($|A|$). A path, denoted as P, from a node $w \in V$ to a node $v \in V$ in network G is defined as a sequence of nodes $P: (w = i_1,\ i_2,\ \ldots,\ i_l = v)$, where l is equal to or greater than 1, and each consecutive pair of nodes $(i_k,\ i_{k+1})$ is an arc from A for every $k = 1,\ 2,\ \ldots,\ l-1$.

Combinatorial optimization is a special class of mathematical program that consists of finding an optimal object among a finite set of specifically structured objects. Some of the most prominent problems of this class are shortest path (SP) problems, maximum reliability path (MRP) problems, and maximum capacity path (MCP) problems/widest path problem (WPP). In these problems, the goal is to find an optimal path from an origin to a destination under a special objective function as follows:

1. $\min\limits_{P\in\mathbb{P}} \sum\limits_{(i,j)\in P} l_{ij}$ for SP problems.
2. $\max\limits_{P\in\mathbb{P}} \prod\limits_{(i,j)\in P} p_{ij}$ for MRP problems.
3. $\max\limits_{P\in\mathbb{P}} \min\limits_{(i,j)\in P} u_{ij}$ for WPP.

where the set $\mathbb{P}$ consists of all paths from the origin to the destination. In this context, the parameters l_{ij}, p_{ij}, and u_{ij} represent the length, reliability, and capacity of arc (i, j) respectively. Fortunately, these problems are tractable and there exist polynomial time algorithms to solve them. For instance, the shortest path problem can be solved using Dijkstra's algorithm with a Fibonacci heap implementation [1], which has a complexity of $O(m + n\log(n))$ if the lengths, l_{ij}, are nonnegative. In case the lengths are negative, the best-known algorithm is a FIFO implementation of the Bellman–Ford algorithm [2,3], with a complexity of $O(mn)$, where n and m represent the number of nodes and arcs, respectively. The maximum reliability path problem can be transformed into a shortest path problem by defining l_{ij} as $-\log(p_{ij})$ for every arc (i, j). Consequently, it can be solved similarly to the shortest path problem, especially when p_{ij} is less than 1. Additionally, both the maximum reliability path problem and the widest path problem can be solved directly by modifying the shortest path algorithms. This is because they share similar optimality conditions with the shortest path problem (refer to [4] for more details). However, the best-known algorithm for solving the widest path problem in an undirected network does not rely on this concept. Instead, it employs a recursive algorithm with a linear complexity of $O(m)$ [5]. The maximum capacity problem has many applications. The service vehicles and emergency vehicles should use such a path when returning from a service call to the base [6]. Also, the maximum capacity problem could be used to deal with computation problems where only comparison of the graph's arcs values is permitted [7,8].

The WPP finds its application in various domains. For instance, let us consider a network that represents connections between routers on the Internet. In this context, each arc in the network denotes the bandwidth of the corresponding connection between two routers. With the WPP, the objective is to discover the path between two Internet nodes that offers the highest possible bandwidth. This network routing problem is well-known in the field. Apart from being a fundamental network routing problem, the WPP also plays a crucial role in other areas. One noteworthy application is within the Schulze method, which is utilized for determining the winner of a multiway election [9]. In this method, the WPP aids in resolving ties and determining the strongest path among multiple alternatives. Additionally, the WPP finds application in digital compositing, wherein it assists in combining multiple images or video layers into a final composite image or sequence [10]. By identifying the path with the maximum capacity, the compositing process can ensure the most efficient allocation of computational resources. Moreover, the problem contributes to metabolic pathway analysis, which involves studying chemical reactions within biological systems. In this context, the WPP aids in understanding the flow of metabolites through various pathways and identifying the most influential pathways in terms of capacity [11]. In summary, the WPP has extensive applications ranging from network routing on the Internet, multiway election methods, and digital compositing, to metabolic pathway analysis. Its capability to identify and utilize paths with the highest capacity proves valuable across these diverse domains.

In this paper, a new combinatorial optimization problem called the generalized widest path (GWPP) problem is introduced. It is a more intricate version of the problem that involves finding a directed path from a given source node s to a given sink node t, with the minimum loss among all available directed paths from s to t [12]. To our knowledge, this is the only paper in the literature that studies st-paths in the gain/loss case. The GWPP is defined on a network where each arc is characterized by two attributes: capacity and loss/gain factor.

The capacity of an arc represents the maximum flow value that can be transmitted through it. On the other hand, the loss/gain factor of an arc indicates the flow value that arrives at the tail node when one unit of flow is sent through the arc. The objective of the GWPP is to find a path that is capable of transmitting the maximum flow while considering the loss/gain factors.

This problem is inspired by an extension of the maximum flow problems [13,14] that incorporates loss/gain factors, known as the generalized maximum flow problem. There-

fore, the algorithms developed for solving the GWPP can also be utilized as subroutines for addressing generalized maximum flow problems.

Moreover, the GWPP can be viewed as an extension of the maximum reliability path and widest path problems. It becomes equivalent to the MRP problem when capacities are infinite and transforms into the WPP when the loss/gain factors are equal to 1. Thus, the GWPP upon the scope of both MRP and the WPPs, encompassing their characteristics and generalizations.

In this paper, we introduce the GWPP—a novel combinatorial optimization problem that extends the concept of the WPP by incorporating loss/gain factors. The algorithms developed for the GWPP can be utilized for generalized maximum flow problems, making it a versatile and applicable problem in various contexts.

The rest of this paper is organized as follows. In Section 2, we provide the necessary background information and definitions to lay the foundation for the research work. Section 3 clearly defines the research problem and outlines its significance. Section 4 describes, in detail, the first two proposed algorithms to solve the GWPP in the case of loss factors. In Section 5, the more general case of loss/gain factors is studied, and an algorithm is presented to solve the GWPP in this general case. Section 6 presents the experiments conducted to validate and evaluate the proposed algorithms. Finally, Section 7 summarizes the main findings of the paper.

The current paper is an extension of the paper presented and published in the proceedings of the 13th International Conference on Operations Research and Enterprise Systems (ICORES 2024) [15]. In the current paper, the general case (with loss/gain factors) is studied, and a strongly polynomial algorithm is introduced to solve the GWPP in the general case (see Section 5). Also, in order to illustrate the functionality of the proposed algorithms, the iterations of each of the three algorithms are presented for a given network (see Sections 4 and 5). The experiments from Section 6 were extended for Algorithm 3 [15] as well.

2. Preliminaries

For the sake of simplicity, from now on, we refer to a path from s to t as an "st-path".

The capacity of an st-path P is denoted by $u(P)$ and is given by the minimum of its capacities, that is,

$$u(P) = \min\{u(a) | a \in P\}.$$

In the network G is intended to find an st-path $\widetilde{P}$ with the maximum capacity among all st-paths:

$$u\left(\widetilde{P}\right) = \max\{u(P) | P \text{ is an } s-t \text{ path}\}.$$

This problem is also called the widest path problem, the bottleneck shortest path problem, and the max–min path problem in the literature.

3. Problem Formulation

In this section, we will delve into the problem of the generalized widest path. Let us formally define the problem considering a connected and directed network denoted as $G(V,\ A,\ u,\ p)$. Here, p is the loss/gain factor parameter.

For each arc $(i,\ j)\ \in\ A$, there are two key parameters associated with it. The first one is the capacity, denoted as u_{ij}, which represents the maximum amount of flow that can be sent along the arc. The second parameter p_{ij} is the loss factor if it lies in the interval $(0, 1]$, or it is the gain factor if $p_{ij} \in (1, +\infty)$. The loss factor captures physical transformations such as evaporation, energy dissipation, breeding, theft, or interest rates [16]. The loss/gain factor could be the exchange rate of two different currencies.

Considering the flow along the arcs, if x_{ij} units of flow enter arc $(i,\ j)$, $p_{ij}x_{ij}$ units of flow are actually delivered to node j. If $p_{ij} \in (0, 1]$, then this implies that $\left(1 - p_{ij}\right)x_{ij}$ units of flow are absorbed or lost along the arc due to the specified loss factor. If $p_{ij} \in (1, +\infty)$, then this means that flow is increased by $\left(p_{ij} - 1\right)x_{ij}$ units.

The aim of the generalized widest path problem is to find an st-path that enables the transmission of the maximum possible flow while taking the loss/gain factors into consideration.

To formulate this problem in a precise manner, we introduce the following variables:

- x_{ij}, representing the flow entering arc $(i,\ j)$.
- y_{ij}, a binary variable that determines whether or not arc $(i,\ j)$ carries a positive flow (1 if it does, 0 otherwise).

Now, we can express the GWPP as a mixed zero–one linear programming model, which can be stated as follows:

$$\max z = v_t \tag{1a}$$

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} p_{ji}x_{ji} = \begin{cases} v_s & i = s, \\ 0 & i \neq s,t, \\ -v_t & i = t, \end{cases} \forall i \in V, \tag{1b}$$

$$\sum_{j:(i,j)\in A} y_{ij} \leq 1,\ \forall i \in V\backslash\{t\}, \tag{1c}$$

$$0 \leq x_{ij} \leq u_{ij}y_{ij},\ \forall (i,j) \in A. \tag{1d}$$

Let us break its points down into separate parts:

1. The variables v_s and v_t represent the flow leaving the source node s and the flow entering the sink node t, respectively.
2. Constraints (1b) and (1d) correspond to the balanced flow constraint and the bound constraints commonly found in maximum flow problems [4].
3. Constraint (1c) ensures that, at most, one outgoing arc from any node is capable of sending flow. This constraint guarantees that flow is sent only along a single st-path.
4. The formulation (1) of the GWPP closely resembles that of generalized maximum flow problems, with the added inclusion of zero-one variables y_{ij} and the constraint (1c) [4].

Remark 1. *While we have assumed that $p_{ij} \leq 1$, constraint (1d) does not account for the possibility of a flow increment on arc (i,j) when $p_{ij} > 1$. In such cases, it should be written as $0 \leq max\{x_{ij}, p_{ij}x_{ij}\} \leq u_{ij}y_{ij}$. To handle this situation without loss of generality, we can redefine the capacity of arc (i,j) as $min\{u_{ij}, u_{ij}/p_{ij}\}$.*

4. Algorithms for the Case of Losses on Arcs

This section focuses on the development of two algorithms to solve the GWPP in polynomial time. This provides evidence that the problem is tractable, similar to the WPP, SP, and MRP problems.

We start with a simple observation: if we send the maximum flow along a path, then at least one of its arcs will be saturated. The capacity of this saturated arc determines the flow value along the path. Let $(i_P,\ j_P)$ be the last saturated arc in an st-path $P = s - \cdots - i_P - j_P - \cdots - k - t$. The flow value along path P is then equal to $u_{i_Pj_P} \times (p_{i_Pj_P} \times \ldots \times p_{kt})$. An interesting insight is that if we remove the arc $(i_P,\ j_P)$ from the network and add a new arc $(s,\ j_P)$ with capacity $u_{sj_P} = u_{i_Pj_P}$, loss factor $p_{sj_P} = p_{i_Pj_P}$, then we can send the same flow value along the new path $s - j_P - \cdots - k - t$. This simple idea leads to a polynomial time algorithm for solving problem (1).

In the first algorithm, disregarding arc capacities, we aim to find a maximum reliability path from s to t, which is a path P where the value of the product $\prod_{(i,j)\in P} p_{ij}$ is maximized. This can be achieved by assigning arc lengths $l_{ij} = -log(p_{ij})$ and finding the shortest st-path based on these arc lengths. Let P be the shortest path obtained. Then, we identify the last arc $(i_P,\ j_P)$ in P that would become saturated if we were to send the maximum flow along P. We remove arc $(i_P,\ j_P)$ from the network and introduce an artificial arc $(s,\ j_P)$ with a loss factor $p_{sj_P} = p_{i_Pj_P}$, capacity $u_{sj_P} = u_{i_Pj_P}$, and a weight of $-\log(u_{i_Pj_P}p_{i_Pj_P})$. We

repeat this process until we find a path P, in which the last saturated arc is one of the artificial arcs.

Considering the unique characteristics of our algorithm, it is noteworthy to underscore that the negative weights exclusively pertain to arcs emanating from the source node s. This crucial distinction enables the seamless application of Dijkstra's algorithm, as its efficacy is contingent upon the absence of negative cycles within the graph. To further streamline the application of Dijkstra's algorithm and eliminate negative arcs altogether, we propose a judicious adjustment to the arc weights. Specifically, we suggest augmenting all arcs with the minimum value among the arcs originating from s, denoted as $\min(s,\ j)$ for the pair $(s,\ j)$. This augmentation ensures the absence of negative weights in the entire graph, rendering it amenable to Dijkstra's algorithm without any reservations.

Subsequently, upon identifying the optimal path and obtaining the computed result, we advocate for subtracting the added value, which is representative of the minimum value among the source-emerging arcs. This corrective measure guarantees the restoration of the original, unaltered values on the optimal path while harnessing the benefits of an adjusted graph conducive to the successful application of Dijkstra's algorithm.

It is important to note that the optimal value of problem (1) is equal to the maximum flow along the last path found by the algorithm. To obtain the optimal path, we need to save the segment of path P from s to i_P whenever $(i_P,\ j_P)$ is removed and $(s,\ j_P)$ is added. This can be accomplished by introducing an additional parameter P_{sj_P} for each artificial arc. Therefore, if the algorithm finds path P in the last iteration, the optimal solution will be a path that includes arcs from P_{sj_P} and P, excluding $(s,\ j_P)$.

Algorithm 1 provides a formal description of our first algorithm. Since an arc is removed in each iteration, the number of iterations is, at most, equal to m (the total number of arcs). Algorithm 1 runs, at most, m times classical Dijkstra's algorithm in the modified network (by applying *–log* to the initial values). As a result, we can conclude the following.

Theorem 1. *The complexity of Algorithm* 1 *is* $O(mS(n,m))$*, in which* $S(m,n)$ *is the complexity of finding the shortest path in the network.*

Algorithm 1 (Alg1)

Input: An instance of the generalized WPP
Output: An optimal path

for $(i,j) \in A$:
  if $i == s$:
    Set $\bar{l}_{ij} = -\log p_{ij}u_{ij}$ and $P_{ij} = (i,j)$
  else:
    Set $\bar{l}_{ij} = -\log p_{ij}$ and $P_{ij} = \varnothing$
while True:
  Find a shortest path P with respect to $\bar{l}_{ij}$
  if $P_{sj_P} \neq \varnothing$:
  The optimal path is $P_{sj_P} \cup P \backslash \{(s, j_P)\}$
  else:
    Find the last arc (i_P, j_P) of P to be saturated.
    Remove (i_P, j_P).
    Add an artificial arc (s, j_P)
    Set $\bar{l}_{sj_P} = -\log(u_{i_Pj_P}p_{i_Pj_P})$

Algorithm 1 iterations (Figures 1–9 illustrate the iterations of the algorithm):

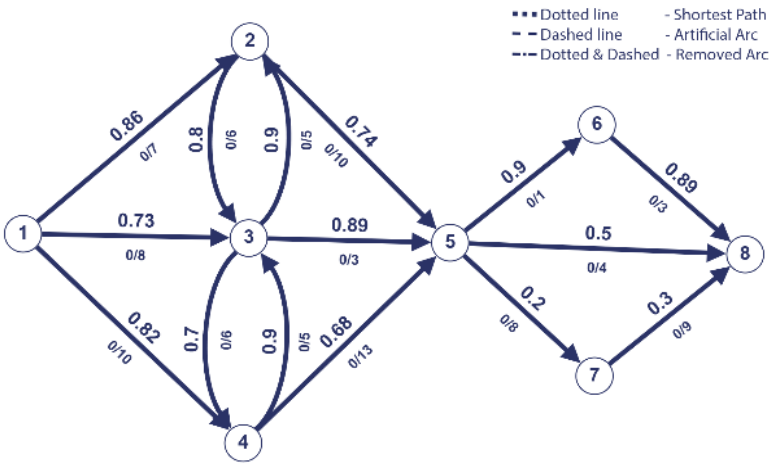

Figure 1. Initial graph.

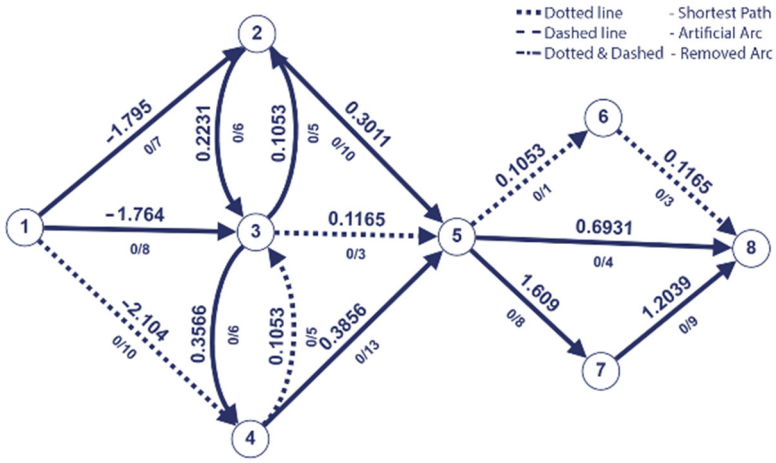

Figure 2. Transformation of problem (1) into minimization problem.

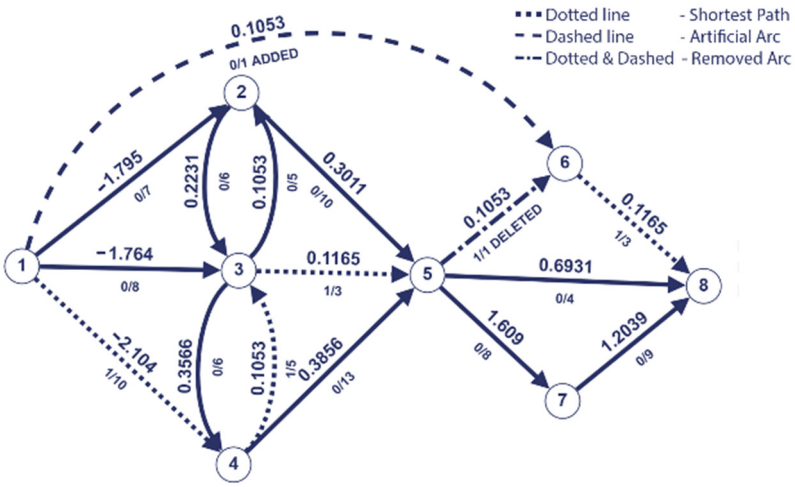

Figure 3. First iteration of the shortest path.

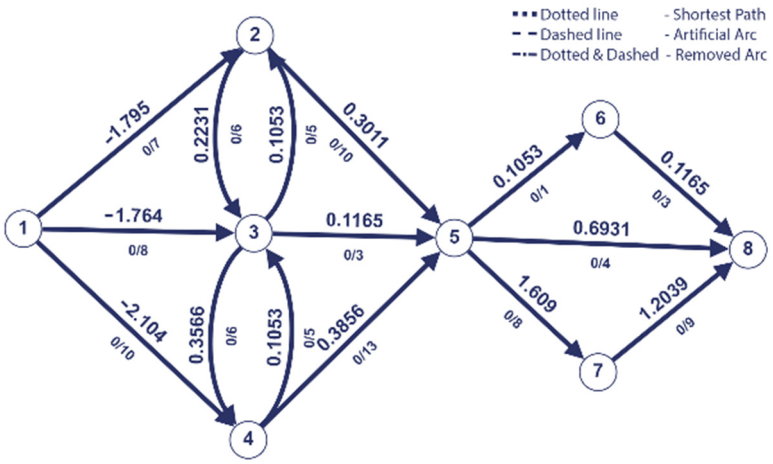

Figure 4. Remove the last saturated arc and add artificial arc.

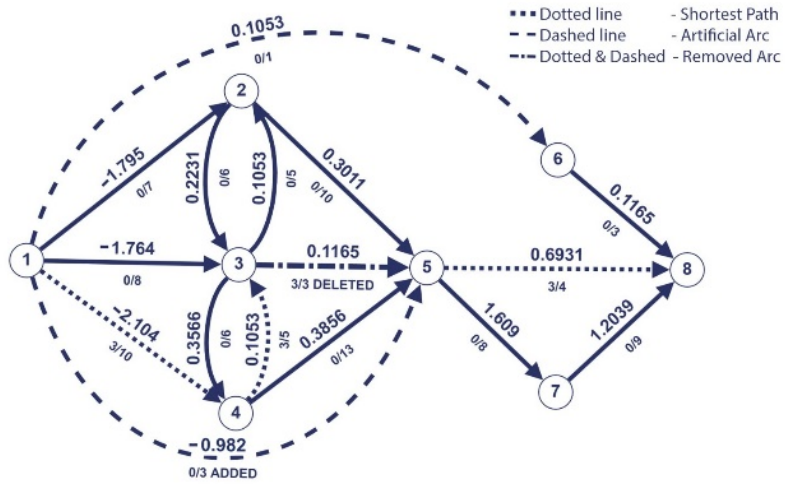

Figure 5. Second iteration of Dijkstra's algorithm.

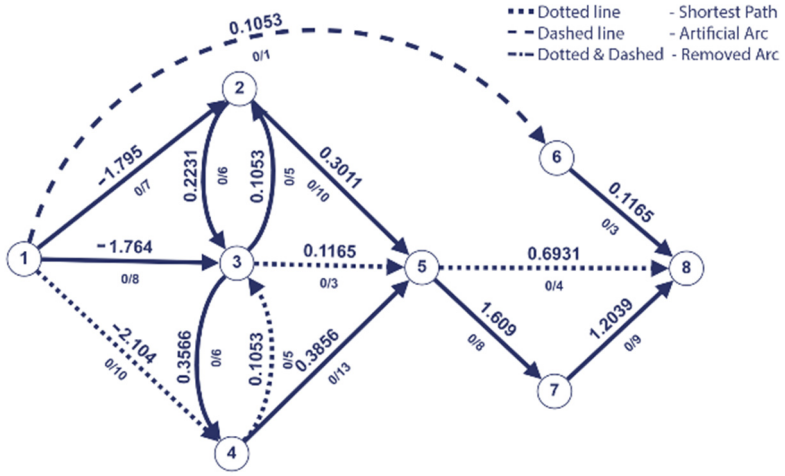

Figure 6. The last saturated arc is removed and add artificial arc.

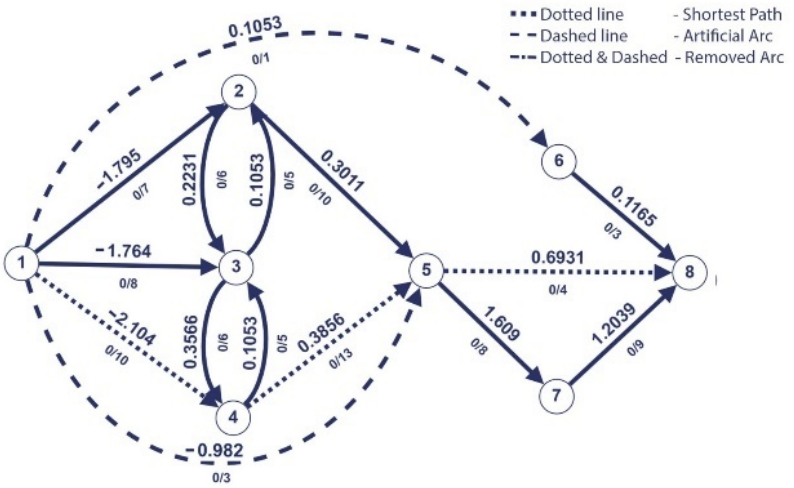

Figure 7. Third iteration of Dijkstra's algorithm.

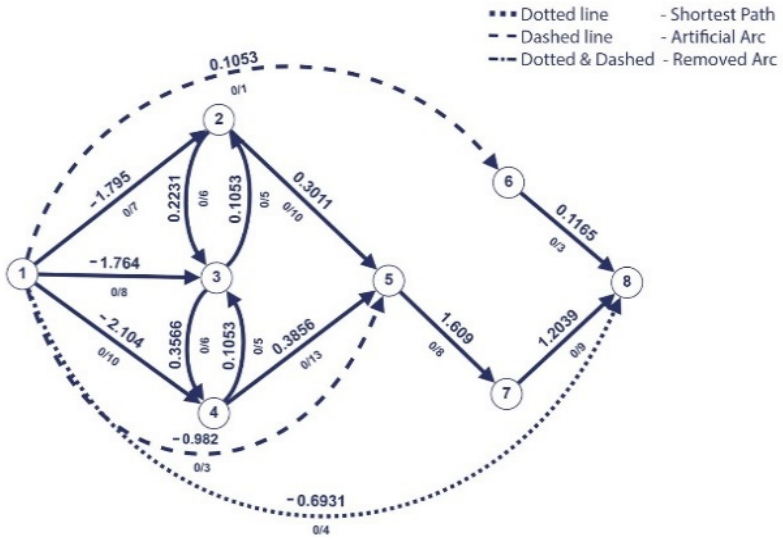

Figure 8. Remove the last saturated arc and add artificial arc.

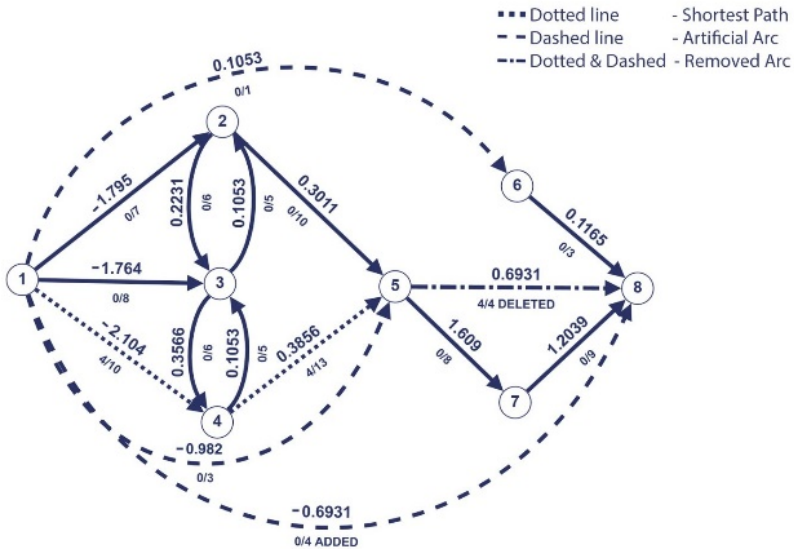

Figure 9. Last iteration of Dijkstra's algorithm.

We observe that our graph example contains arcs with negative weights. However, this does not pose a problem as the negative arcs will always exist from the source node. This particularity ensures that Dijkstra's algorithm performs well.

We applied Dijkstra's algorithm to find the shortest path from source 1 to the sink 8. The resulting path is marked with dotted line and is P = (1,4,3,5,6,8).

The last saturated arc $(5_P,\ 6_P)$ on the shortest path P is eliminated, and the arc (s,6) is added with the capacity of $(5_P,\ 6_P)$ and the weight $\bar{l}_{sj_P} = -\log(u_{i_Pj_P} p_{i_Pj_P})$ =0.1503.

The resulted path on the second iteration is P = (1,4,3,5,8).

On the new shortest path found, after we push the maximum flow through it, we find that the arc $(3_P,\ 5_P)$ is last saturated. So, it is removed, and the arc (1,5) is added with the same capacity of 3 and the new weight of -0.982.

The resulted path on the third iteration is P = (1,4,5,8).

In the third iteration, the arc $(5_P,\ 8_P)$ is found to be the last saturated arc. An artificial arc (1,8) with the capacity 4 and weight -0.6931 is added.

The last saturated arc is one of the artificial arcs; therefore, we stop here. In this scenario, we applied Dijkstra's algorithm fourth times. After we apply $P_{sj_P} \cup P \backslash \{(s, j_P)\}$, which, in our scenario, is $P_{sj_P} = (1, 4, 5)$ and $P \backslash \{(s, j_P)\}$ = (5,8), we obtain the optimal path (1,4,5,8) as can be seen in Figure 10.

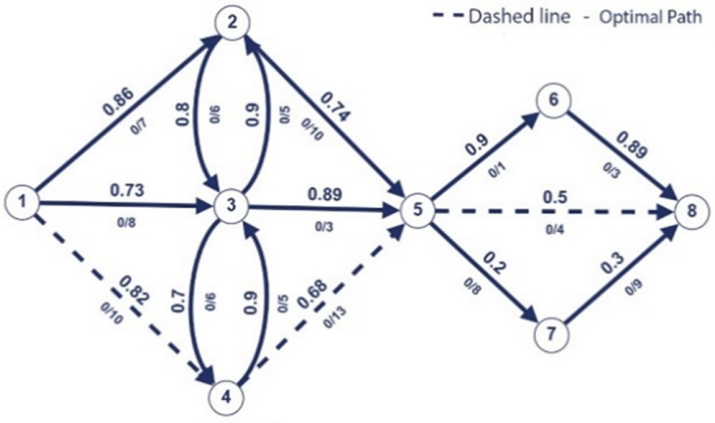

Figure 10. Optimal path.

In the followings, we discuss optimality conditions for problem (1) and present an algorithm with a time complexity of $O(S(m, n))$, which improves the complexity of Algorithm 1 by a factor of m.

To begin, we introduce a label $d(j)$ for each node $j \in V$. During intermediate stages of computation, the label $d(j)$ serves as an estimate (or an upper bound) of the maximum flow sent from the source node s to node j along a single path. At the termination of the algorithm, the label $d(j)$ represents the optimal value of problem (1). Our objective is to establish necessary and sufficient conditions for a set of labels to accurately represent the maximum flow.

Let $d(j)$ denote the value of the maximum flow sent from the source node to node j (where we set $d(s) = +\infty$). In order for the labels to be optimal, they must satisfy the following necessary optimality conditions:

- Constraint (1c): For each node $j \neq s$, there exists, at most, one outgoing arc with positive flow. This condition ensures that the flow is sent only along a single *st*-path.
- Capacity Constraint: For each arc (i, j), the flow through the arc must not exceed its capacity. Mathematically, this can be written as $f_{ij} \leq u_{ij}$, where f_{ij} represents the flow on arc (i, j) and u_{ij} represents the capacity of arc (i, j).
- Flow Conservation: The flow conservation principle must be satisfied at every node (except the source and sink nodes). For any node $j \neq s$ and $j \neq t$, the sum of incoming flows must equal the sum of outgoing flows. Mathematically, this can be expressed as $\sum_{(i,j)\in A} f_{ij} - \sum_{(j,k)\in A} f_{jk} = 0$.

- Optimality Condition: For each node $j \neq s$, the label $d(j)$ represents the maximum flow sent from the source node s to node j. Therefore, we have $d(j) = \sum_{(i,j)\in A} f_{ij} - \sum_{(j,k)\in A} f_{jk}$, where f_{ij} represents the flow on arc (i,j) and f_{jk} represents the flow on arc (j,k).

By satisfying these necessary optimality conditions, we can ensure that the labels $d(j)$ accurately represent the maximum flow in the network.

If the labels are optimal, they must satisfy the following necessary optimality conditions:

$$d_j \geq p_{ij}\min\{u_{ij}, d_i\}.$$

This is an extension of the optimality conditions of both the WPP and MRP problem. On the optimal path, the inequality is satisfied in the equality form. It states that the label of node j is either $p_{ij}d_i$ or $p_{ij}u_{ij}$. In the case that the flow value arriving at node i is less than u_{ij}, (namely, $d_i < u_{ij}$), this arc is not saturated and consequently $d_j = p_{ij}d_i$. In the other case, (i,j) is saturated, sending more flow than its capacity. So, $d_j = p_{ij}u_{ij}$ in this case.

Since this optimality condition is similar to that of the SP problem, we can apply the concept of Dijkstra's algorithm to solve problem 1. This is presented in Algorithm 2.

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

Proof of Theorem 2. Considering the provided information, we can deduce that in the *while* loop, each arc is checked only once. As a result, the number of iterations executed by the two last lines of the loop is, at most, $O(m)$, which is also less than or equal to $O(n^2)$ considering the worst-case scenario. □

Conversely, the node selection process, where the node with the minimum label is chosen, requires $O(n)$ time in each iteration. Taking into account that the number of iterations is $O(n^2)$, we can conclude that the most time-consuming operation in this algorithm is the node selection, which takes $O(n^2)$ time. Algorithm 2 is derived from Dijkstra's algorithm by replacing the classical test with $d_j < p_{ij}\min\{u_{ij}, d_i\}$; then, if necessarily, the update $d_j = p_{ij}\min\{u_{ij}, d_i\}$ is performed.

Algorithm 2 (Alg. 2)

Input: An instance of the generalized WPP
Output: An optimal path

for $i \in V$:
Set $d(i) = 0$
Set $d_s = +\infty$
Set $S_0 = \{\}$; $\overline{S} = V$
while $|S| < n$:
  Let $i \in S$ be a node for which
    $d_i = \max\left\{d_j : j \in \overline{S}\right\}$;
  $\overline{S} = \overline{S}\backslash\{i\}$;
  for each $j \in V : (i,j) \in A$:
    if $d_j < p_{ij}\min\left\{u_{ij}, d_i\right\}$:
      Update $d_j = p_{ij}\min\left\{u_{ij}, d_i\right\}$
      Update $S = S\ U\ \{i\}$;

We can also use the available implementations of Dijkstra's algorithm for the complexity improvement of Algorithm 2. For example, the Fibonacci heap implementation reduces the complexity to $O(m + n\log n)$.

Algorithm 2 iterations:

We will apply Algorithm 2 for the graph from Figure 1. We have the following data structures: d (distance vector), Pred. (predecessor vector), and S (priority queue or a Fibonacci heap, depending on the implementation). It is important to note that all structures are starting from index 1 in this example.

Upon examination of the predecessor vector Pred. represented by [−1, 1, 1, 1, 4, 5, 5, 5], it is obvious that the optimal path is (1, 4, 5, 8). So, both the first and second algorithms yield identical optimal paths (see Figure 10 and Table 1).

Table 1. Algorithm 2 iterations.

No. itr.	Structure	Structure Data
1	D	[inf, 0, 0, 0, 0, 0, 0, 0]
	S	[1]
	Pred.	[−1, −1, −1, −1, −1, −1, −1, −1]
2	D	[inf, 6.02, 5.84, 8.2, 0, 0, 0, 0]
	S	[2, 3, 4]
	Pred.	[−1, 1, 1, 1, −1, −1, −1, −1]
3	D	[inf, 6.02, 5.84, 8.2, 5.44, 0, 0, 0]
	S	[2, 3, 5]
	Pred.	[−1, 1, 1, 1, 4, −1, −1, −1]
4	D	[inf, 6.02, 5.84, 8.2, 5.44, 0, 0, 0]
	S	[3, 5]
	Pred.	[−1, 1, 1, 1, 4, −1, −1, −1]
5	D	[inf, 6.02, 5.84, 8.2, 5.44, 0, 0, 0]
	S	[5]
	Pred.	[−1, 1, 1, 1, 4, −1, −1, −1]
6	d	[inf, 6.02, 5.84, 8.2, 5.44, 0.9, 1, 2]
	S	[6, 7, 8]
	Pred.	[−1, 1, 1, 1, 4, 5, 5, 5]
7	D	[inf, 6.02, 5.84, 8.2, 5.44, 0.9, 1, 2]
	S	[6, 7]
	Pred.	[−1, 1, 1, 1, 4, 5, 5, 5]
8	D	[inf, 6.02, 5.84, 8.2, 5.44, 0.9, 1, 2]
	S	[6]
	Pred.	[−1, 1, 1, 1, 4, 5, 5, 5]
9	D	[inf, 6.02, 5.84, 8.2, 5.44, 0.9, 1, 2]
	S	[EMPTY]
	Pred.	[−1, 1, 1, 1, 4, 5, 5, 5]

5. Gain/Loss Case of the GWPP

Since the GWPP is reduced to a minimum path search problem in a network with non-negative values on arcs (except for the arcs starting with the source node), Dijkstra's algorithm was adapted for solving the GWPP. The same optimality conditions may apply to problems involving gain factors (leading to arcs with negative values in the modified network). Therefore, Bellman–Ford's algorithm can be adapted to address the case when the arcs have gain/loss factors. We used the shortest path faster algorithm (SPFA) implementation of Bellman–Ford [17]. The algorithm iteratively relaxes edges in the graph and updates the optimal path estimates until no further updates are possible. This algorithm is described in Algorithm 3. The correctness of Algorithm 3 is akin to that of the well-known Bellman–Ford algorithm (the modifications performed for the classical Bellman–Ford's algorithm are similar to those conducted for Dijkstra's algorithm). For a proof, please refer to [4].

Algorithm 3 (Alg. 3)

Input: An instance of the GWPP with gain/loss factors
Output: An optimal path

for $i \in V$:
  Set $d_{\mathrm{i}} = 0,\ pred_i = -1,\ S_i = false$
Set $d_{\mathrm{s}} = +\infty$, $Q = \{s\}$;
while Q has elements:
  $i = pool\ Q$;
  $S_i = false$;
  for $(i, j) \in A$:
    if $d_j < p_{ij}\min\left\{u_{ij}, d_i\right\}$:
      Update $d_j = p_{ij}\min\left\{u_{ij}, d_i\right\}$.
      Set $pred_j = i$.
      if $S_j == false$:
        $Q = Q\ U\ \{j\}$;
        $S_j = true$;
Restore the optimal path by the *Pred* indices.

Algorithm 3 iterations:

No. itr.	Selected Edge	Structure	Structure Data
0—init.	-	Q	[1]
		S	[true,false,false,false,false,false,false,false]
		d	[inf, 0, 0, 0, 0, 0, 0, 0]
		Pred.	[−1, −1, −1, −1, −1, −1, −1, −1]
1	(1,2)	Q	[2]
		S	[false,true,false,false,false,false,false,false]
		d	[inf, 6.02, 0, 0, 0, 0, 0, 0]
		Pred.	[−1, 1, −1, −1, −1, −1, −1, −1]
2	(1,3)	Q	[2, 3]
		S	[false,true,true,false,false,false,false,false]
		d	[inf, 6.02, 5.84, 0, 0, 0, 0, 0]
		Pred.	[−1, 1, 1, −1, −1, −1, −1, −1]
3	(1,4)	Q	[2, 3, 4]
		S	[false,true,true,true,false,false,false,false]
		d	[inf, 6.02, 5.84, 8.20, 0, 0, 0, 0]
		Pred.	[−1, 1, 1, 1, −1, −1, −1, −1]
3	(2,3)	Q	[3, 4]
		S	[false,false,true,true,false,false,false,false]
		d, Pred.	No change
4	(2,5)	Q	[3, 4, 5]
		S	[false,false,true,true,true,false,false,false]
		d	[inf, 6.02, 5.84, 8.20, 4.45, 0, 0, 0]
		Pred.	[−1, 1, 1, 1, 2, −1, −1, −1]
5	(3,2)	Q	[4, 5]
		S	[false,false,false,true,true,false,false,false]
		d, Pred.	No change
5	(3,4)	Q, S, d, Pred.	No change
6	(3,5)	Q, S, d, Pred.	No change
7	(4,3)	Q	[5]
		S	[false,false,false,false,true,false,false,false]
		d, Pred.	No change

8	(4,5)	Q, S	No change
		d	[inf, 6.02, 5.84, 8.20, 5.44, 0, 0, 0]
		Pred.	[−1, 1, 1, 1, 4, −1, −1, −1]
9	(5,6)	Q	[6]
		S	[false,false,false,false,false,true,false,false]
		d	[inf, 6.02, 5.84, 8.20, 5.44, 0.90, 0, 0]
		Pred.	[−1, 1, 1, 1, 4, 5, −1, −1]
10	(5,7)	Q	[6, 7]
		S	[false,false,false,false,false,true,true,false]
		d	[inf, 6.02, 5.84, 8.20, 5.44, 0.90, 1.0, 0]
		Pred.	[−1, 1, 1, 1, 4, 5, 5, −1]
11	(5,6)	Q	[6, 7, 8]
		S	[false,false,false,false,false,true,true,true]
		d	[inf, 6.02, 5.84, 8.20, 5.44, 0.90, 1.0, 2.0]
		Pred.	[−1, 1, 1, 1, 4, 5, 5, 5]
12	(6,8)	Q	[7, 8]
		S	[false,false,false,false,false,false,true,true]
		d, Pred.	No change
13	(7,8)	Q	[8]
		S	[false,false,false,false,false,false,false,true]
		d, Pred.	No change
14	(8,-) No neighbors	Q	[EMPTY]
		S	[false,false,false,false,false,false,false,false]
		d, Pred.	No change

The complexity of the algorithm is stated in the following theorem.

Theorem 3. *Algorithm 3 solves the GWPP in the presence of loss and gain factors in $O(nm)$ time.*

Proof of Theorem 3. See [4]. □

While Algorithm 3 is less efficient than Algorithm 2, its ability to handle gain factors makes it a valuable tool in the more general case of gain or loss factors attached to the arcs.

6. Experiments and Discussions

Since the problem studied in the paper is new, the (three) algorithms proposed in the paper were compared with each other. The results of the comparison can be seen below.

Based on the findings presented in Theorems 1 and 2, it is obvious that Algorithm 2 outperforms Algorithm 1 in terms of speed. However, Algorithm 1 offers an advantage in that it can be effectively parallelized on GPUs by utilizing classical Dijkstra's algorithm [18] in a sequential manner. So, the implementation of CUDA-based Dijkstra's algorithm resulted in a noteworthy acceleration of Algorithm 1.

The CUDA version exhibited significantly improved performance compared to Algorithm 2, as shown in Figure 11. Particularly for larger network instances with 10,000 nodes or more, the CUDA implementation of Algorithm 1 was up to 25.7 times faster than that of Algorithm 2.

As expected, Algorithm 2 is faster than Algorithm 3 (see Figure 12). The experiments were performed on large networks (see Table 2). They showed that Algorithm 2 runs up to 1.64 times faster than Algorithm 3 (see Table 3).

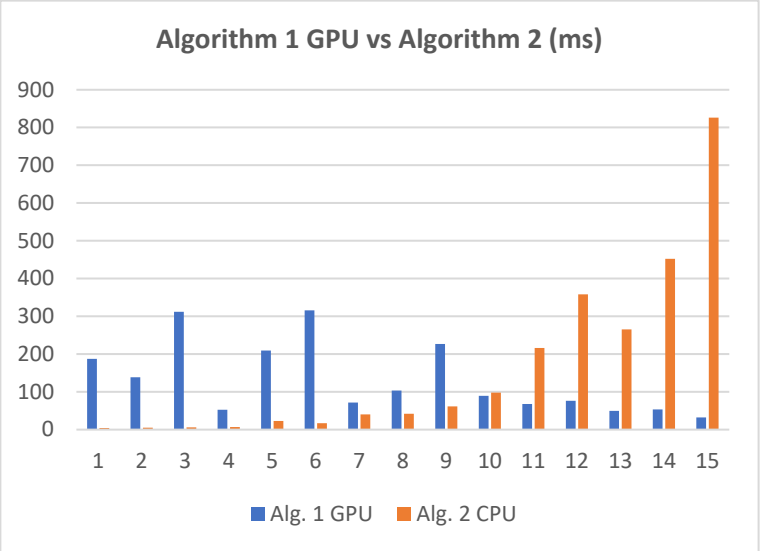

Figure 11. Running times (ms) comparison between Algorithm 1 GPU and Algorithm 2 CPU.

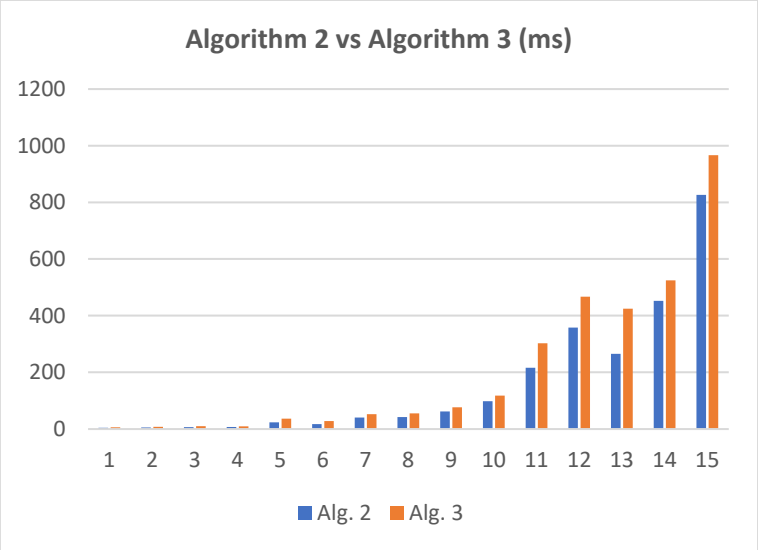

Figure 12. Running times (ms) comparison between Algorithm 2 and Algorithm 3.

We used different instances of random generated networks with the number of nodes varying from 1000 to 25,000 (see Table 1). The networks were generated using the network random generator from [19]. It is imperative to acknowledge that throughout various experimental analyses, the graph's density played an important role and was calibrated using the Erdős–Rényi probability distribution [20]. The Erdős–Rényi variable, a continuous parameter spanning the interval [0, 1], played a pivotal role in these experimental investigations.

In this context, it is essential to elucidate that the Erdős–Rényi variable assumes a value of 0 when the graph lacks any new arcs, signifying minimal density. Conversely, a value of 1 denotes the graph's attainment of maximum density, highlighting the comprehensive spectrum of density exploration in our experimental framework.

Table 2. Network generator parameters used for experiments.

No. of Nodes	No. of Instances	No. of Paths	No. of Cycles	Erdős–Rényi Prob.	No.
1000	10,000	500	100	0.5	1
		500	100	0.7	2
		500	100	0.9	3
		1000	100	0.1	4
		1000	250	0.15	5
		1000	500	0.6	6
5000	100	2500	1000	0.1	7
		2500	1000	0.2	8
		2500	1000	0.3	9
		5000	2500	0.15	10
		5000	2500	0.3	11
		5000	2500	0.5	12
15,000	3	7500	1000	0.15	13
20,000	2	7500	1000	0.15	14
25,000	1	8000	1500	0.15	15

Table 3. Running times (ms) comparison between algorithms.

No.	Algorithm 1 CPU	Algorithm 1 GPU	Algorithm 2	Algorithm 3	Algorithm 1 GPU vs. Algorithm 2 (Times Faster)	Algorithm 2 vs. Algorithm 3 (Times Faster)
1	165.75	187.30	3.98	5.55	0.02	1.39
2	121.1	138.66	5.02	7.52	0.04	1.49
3	188.96	311.78	6.06	9.52	0.02	1.57
4	30.22	52.40	6.86	8.90	0.13	1.29
5	120.00	209.34	23.00	36.18	0.10	1.57
6	193.50	315.60	16.92	27.88	0.05	**1.64**
7	183.00	71.52	40.15	52.19	0.56	1.29
8	284.30	103.41	41.7	55.00	0.40	1.31
9	626.6	226.74	61.4	76.45	0.27	1.24
10	671.10	89.21	98.01	117.61	1.10	1.19
11	677.14	67.78	216.00	302.40	3.19	1.40
12	940.20	76.15	358.01	466.6	4.70	1.30
13	1306.07	49.41	265.33	424.52	4.82	1.59
14	2965.04	53.48	452.11	524.44	8.46	1.15
15	3549.10	32.11	826.04	966.46	**25.72**	1.16

The highest differences between algorithms' running speeds are highlighted in Table 3.

The experiments were performed using a PC with an Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz, 24 GB RAM, and an NVIDIA GeForce GTX 1070 TI graphics card. The algorithms were programmed in Visual C++ 2022 under Windows 10.

7. Conclusions

In this paper, a novel combinatorial optimization problem was introduced (GWPP). The main objective is to identify a path that can transmit the maximum flow, taking into account both the capacities and loss/gain factors of the arcs.

The paper presented two strongly polynomial algorithms to address the GWPP in the loss factor case and a third algorithm that can be used to solve the GWPP in the general case of gain/loss factors attached to arcs. The first algorithm has a time complexity of $O(mS(n,m))$, where $S(m,n)$ represents the time complexity of finding the shortest path in the network. On the other hand, the second algorithm has a more efficient time complexity of $O(n^2)$. However, when the first algorithm is implemented on GPUs, it performs substantially faster than the second algorithm, especially for large network instances. Algorithm 3 has a worse time complexity of $O(mn)$ compared to that of Algorithm 2, but Algorithm 3 can deal with the more general case of gain/loss factors. To better understand how

each of the three algorithms works, an example was considered, and the iterations of each algorithm were presented.

The same experiments were performed for all three algorithms, and the performance was compared. Algorithm 1 is considerably slower than Algorithm 2, both programmed on a CPU. However, Algorithm 1 can be easily and efficiently implemented on GPUs, and in this case, it outperforms Algorithm 2. As expected, Algorithm 3 is slower than Algorithm 2.

Author Contributions: Conceptualization, J.T.; methodology, J.T. and A.M.D.; software, M.-L.R.; validation, A.M.D.; formal analysis, J.T.; data curation, M.-L.R. and A.M.D.; writing—original draft preparation, J.T.; writing—review and editing, J.T., M.-L.R. and A.M.D.; visualization, M.-L.R.; supervision, J.T. and A.M.D.; funding acquisition, A.M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
2. Bellman, R. On a routing problem. *Q. Appl. Math.* **1958**, *16*, 87–90. [CrossRef]
3. Ford, L.R., Jr.; Fulkerson, D.R. *A Shortest Chain Algorithm: Flows in Networks*; Princeton University Press: Princeton, NJ, USA, 1962; pp. 130–134.
4. Ahuja, R.K.; Magnanti, T.L.; Orlin, J.B. *Network Flows*; Prentice Hall: Hoboken, NJ, USA, 1993.
5. Punnen, A.P. A linear time algorithm for the maximum capacity path problem. *Eur. J. Oper. Res.* **1991**, *53*, 402–404. [CrossRef]
6. Berman, O.; Handler, G.Y. Optimal Minimax Path of a Single Service Unit on a Network to Nonservice Destinations. *Transp. Sci.* **1987**, *21*, 115–122. [CrossRef]
7. Kaibel, V.; Peinhardt, M.A.F. *On the Bottleneck Shortest Path Problem*; ZIB-Report 06-22; Konrad-Zuse-Zentrum für Informationstechnik Berlin: Berlin, Germany, 2006.
8. Gabow, H.N.; Tarjan, R.E. Algorithms for two bottleneck optimization problems. *J. Algorithms* **1988**, *9*, 411–417. [CrossRef]
9. Schulze, M. A new monotonic, clone-independent, reversal symmetric, and Condorcet-consistent single-winner election method. *Soc. Choice Welf.* **2011**, *36*, 267–303. [CrossRef]
10. Fernandez, E.; Garfinkel, R.; Arbiol, R. Mosaicking of aerial photographic maps via seams defined by bottleneck shortest paths. *Oper. Res.* **1998**, *46*, 293–304. [CrossRef]
11. Ullah, E.; Lee, K.; Hassoun, S. An algorithm for identifying dominant-edge metabolic pathways. In Proceedings of the 2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers, San Jose, CA, USA, 2–5 November 2009; pp. 144–150.
12. Deaconu, A.M.; Ciupala, L.; Spridon, D. Finding minimum loss path in big networks. In Proceedings of the 22nd International Symposium on Parallel and Distributed Computing (ISPDC), Bucharest, Romania, 10–12 July 2023; pp. 39–44.
13. Ford, L.R.; Fulkerson, D.R. Maximal flow through a network. *Can. J. Math.* **1956**, *8*, 399–404. [CrossRef]
14. Edmonds, J.; Karp, R.M. Theoretical improvements in algorithmic efficiency for network flow problems. *JACM* **1972**, *19*, 248–264. [CrossRef]
15. Tayyebi, J.; Rîtan, M.-L.; Deaconu, A.M. Generalized Maximum Capacity Path Problem with Loss Factors. In Proceedings of the 13th International Conference on Operations Research and Enterprise Systems (ICORES 2024), Rome, Italy, 24–26 February 2024; pp. 302–308, ISBN 978-989-758-681-1. ISSN 2184-4372.
16. Tayyebi, J.; Deaconu, A. Inverse generalized maximum flow problems. *Mathematics* **2019**, *7*, 899. [CrossRef]
17. Moore, E.F. The shortest path through a maze. In *Proceedings of the International Symposium on the Theory of Switching*; Harvard University Press: Cambridge, MA, USA, 1959.
18. Ortega-Arranz, H.; Torres, Y.; Llanos, D.R.; Gonzalez-Escribano, A. A new GPU-based approach to the Shortest Path problem. In Proceedings of the 2013 International Conference on High Performance Computing & Simulation (HPCS), Helsinki, Finland, 1–5 July 2013; pp. 505–511.
19. Deaconu, A.M.; Spridon, D. Adaptation of Random Binomial Graphs for Testing Network Flow Problems Algorithms. *Mathematics* **2021**, *9*, 1716. [CrossRef]
20. Erdős, P.; Rényi, A. On Random Graphs. I. *Publ. Math.* **1959**, *6*, 290–297. [CrossRef]