computation

MDPI

Article

Novel GPU-Based Method for the Generalized Maximum Flow Problem

Delia Elena Spridon [1], Adrian Marius Deaconu [1,*] and Javad Tayyebi [2]

[1] Department of Mathematics and Computer Science, Transilvania University of Brașov, 500036 Brașov, Romania
[2] Department of Industrial Engineering, Birjand University of Technology, Birjand 97198-66981, Iran; drjavadtayyebi@gmail.com
* Correspondence: a.deaconu@unitbv.ro

Abstract: This paper investigates the application of a minimum loss path finding algorithm to determine the maximum flow in generalized networks that are characterized by arc losses or gains. In these generalized network flow problems, each arc has not only a defined capacity but also a loss or gain factor, which must be taken into consideration when calculating the maximum achievable flow. This extension of the traditional maximum flow problem requires a more comprehensive approach, where the maximum amount of flow is determined by accounting for additional factors such as costs, varying arc capacities, and the specific loss or gain associated with each arc. This paper extends the classic Ford–Fulkerson algorithm, adapting it to iteratively identify source-to-sink ($s - t$) residual directed paths with minimum cumulative loss and generalized augmenting paths (GAPs), thus enabling the efficient computation of maximum flow in such complex networks. Moreover, to enhance the computational performance of the proposed algorithm, we conducted extensive studies on parallelization techniques using graphics processing units (GPUs). Significant improvements in the algorithm's efficiency and scalability were achieved. The results demonstrate the potential of GPU-accelerated computations in handling real-world applications where generalized network flows with arc losses and gains are prevalent, such as in telecommunications, transportation, or logistics networks.

Keywords: generalized maximum flow; minimum loss path; GPU computing; residual network

check for updates

Academic Editor: Francesco Cauteruccio

Received: 30 October 2024
Revised: 21 January 2025
Accepted: 24 January 2025
Published: 5 February 2025

Citation: Spridon, D.E.; Deaconu, A.M.; Tayyebi, J. Novel GPU-Based Method for the Generalized Maximum Flow Problem. *Computation* **2025**, *13*, 40. https://doi.org/10.3390/computation13020040

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The maximum flow problem is a well-known optimization challenge in graph theory, which focuses on determining the maximum flow that can be transmitted through a network of pipes, channels, or other pathways subject to capacity constraints. In traditional network models, it is assumed that flow is conserved along each arc. The generalized maximum flow problem (GMFP) can be formulated as a linear programming task [1]. Early methods for solving this problem included the augmenting path algorithm and its variations [2]. Tardos and Wayne later introduced the first efficient primal algorithm for the generalized maximum flow problem and extended it to the generalized minimum cost flow problem [3]. A significant advancement was made when a strongly polynomial-time algorithm was introduced for the maximization of the generalized flow, bypassing general linear programming methods based on a novel scaling technique [4]. This problem has wide applications, including modeling various real-world systems like transportation

networks, communication systems, and resource allocation. Additionally, Tayyebi and Deaconu proposed a solution to the inverse generalized flow problem [5].

Determining the maximum flow in networks with gains/losses on arcs is a more advanced concept, relevant to scenarios where flow is diminished by factors such as friction, resistance, degradation, or other reductions in transport capacity. This concept has practical applications in many fields, such as water, electricity, natural gas, transportation, oil, petroleum distribution systems, and telecommunications. In these fields, algorithms designed to compute the maximum flow with arc losses are critical for optimizing resource utilization, minimizing losses, and enhancing the operational efficiency of networks.

In a generalized network, possible losses or gains on arcs are considered. Therefore, for the maximum flow problem in such a generalized network, the loss or gain factors must be taken into consideration when it is computed. This paper is an extension of the paper presented at the International Conference on Applied Computing [6]. A new method to calculate the maximum flow in a generalized network using an algorithm for a minimum loss path is proposed. The classical Ford–Fulkerson algorithm iteratively identifies source-to-sink ($s - t$) paths with minimum cumulative loss, thus enabling the efficient computation of maximum flow in such complex networks. In the conference paper, we considered the particular case when flow generating directed cycles does not appear in the residual network. In the current paper, we extended the algorithm to handle this situation. Moreover, to enhance the computational performance of the proposed algorithm, we conducted extensive studies on parallelization techniques using graphics processing units (GPUs). Significant improvements in the algorithm's efficiency and scalability were achieved. The results demonstrate the potential of GPU-accelerated computations in handling real-world applications where generalized network flows considering arc losses and gains are prevalent, such as in telecommunications, transportation, or logistics networks. GPU computing leverages the parallel processing power of GPUs to perform complex computations more efficiently than traditional CPUs. While CPUs are suitable for sequential tasks, GPUs are designed with thousands of smaller cores capable of handling many operations in parallel. Therefore, they are ideal for tasks that can be executed in parallel, such as matrix computations, deep learning, and large-scale data processing. Compute Unified Device Architecture (CUDA) was developed by NVIDIA and is a parallel computing platform that allows developers to use GPUs for general-purpose computing. CUDA can be used to accelerate a wide range of applications, from scientific simulations to artificial intelligence, resulting in significant performance improvements compared to CPU-only processing. CUDA provides APIs and extensions to common programming languages like C, C++, and Python, making it accessible for developers to integrate GPU acceleration into their applications.

1.1. The Traditional Maximum Flow Problem

The classical maximum flow problem was initially studied by Dantzig and later by Ford and Fulkerson [7,8]. This problem involves finding a method to optimally transport flow from a source node to a sink node while respecting the capacity limits on each arc. Several theoretical components are required to support the approach used in this study.

The primary objective is to transmit the maximum possible flow between two nodes while adhering to the capacity constraints on the arcs. A typical instance of the maximum flow problem is represented by an antisymmetric network, $G = (V, E, s, t, c)$, where V denotes the set of vertices, E represents the set of edges, s is the source node, t is the sink node, and c is a capacity function.

In a network, a flow must satisfy the following boundary constraints:

$$\forall (u, v) \in E : f(u, v) \leq c(u, v)$$

Also, a flow has to satisfy the conservation constraints:

$$\forall v \in V - \{s,t\} : \sum_{u \in V:(u,v) \in E} f(u,v) - \sum_{u \in V:(v,u) \in E} f(v,u) = 0$$
$$\sum_{u \in V:(s,u) \in E} f(s,u) - \sum_{u \in V:(u,s) \in E} f(u,s) = \sum_{u \in V:(u,t) \in E} f(u,t) - \sum_{u \in V:(v,u) \in E} f(t,u)$$

The value of a flow f is determined by the net flow into the sink:

$$|f| = \sum_{u \in V:(u,t) \in E} f(u,t) - \sum_{u \in V:(v,u) \in E} f(t,u)$$

For a network G, the residual network G_f can be defined, where $G_f = \left(V, E_f\right)$, and c_f is the residual capacity function. For example, if $f(u,v) = 3$, $c(u,v) = 5$, $f(v,u) = 1$, $c(v,u) = 4$ and $c_f(u,v) = c(u,v) - f(u,v) + f(v,u) = 3$, $c_f(v,u) = c(v,u) - f(v,u) + f(u,v) = 6$, then the arc (u,v) has 3 units of residual capacity, while the arc (v,u) has 6 units of residual capacity. This means that the flow from the node u to the node v can be increased with 3 units of flow, and from the node v to the node u there is potential flow incrementation of 6 units.

A feasible flow is maximum if and only if there are no augmenting paths in the residual network [8]. If such an augmenting directed path exists in G_f, the flow f can be increased by sending additional flow along this path.

The first method used for solving the maximum flow problem is Ford–Fulkerson, which relies on the concept of augmenting paths. The Ford–Fulkerson algorithm begins with a feasible flow f. Iteratively, a directed path P from s to t is searched in G_f. The minimum capacity, denoted by $r = c_f(P)$, of the edges on the residual path is identified, and the flow along that path is increased by the maximum possible amount. The residual network is then updated along that path according to the following rules:

$$\begin{cases} c_f(u,v) = c(u,v) - r \\ c_f(v,u) = c(v,u) + r \end{cases}$$

This process is repeated until no more augmenting paths can be found in the residual network.

1.2. The Generalized Maximum Flow Problem

The generalized maximum flow problem extends the classical maximum flow problem by allowing the flow to change as it moves through the network. Thus, a gain/loss on each arc of the network is introduced, making it suitable for modeling a wide range of practical scenarios, including economic networks, telecommunications, and logistics.

Like the traditional problem, each arc (u,v) has a capacity $c(u,v)$ that restricts the amount of flow passing through it. However, in the generalized version, each arc (u,v) is also associated with a positive coefficient $\alpha(u,v)$, which is the gain/loss factor. For each unit of flow entering arc (u,v) at node u, only $\alpha(u,v)$ units reach node v. An arc that experiences losses has $\alpha(u,v) < 1$, while an arc with gains has $\alpha(u,v) > 1$.

Over the decades, relatively few algorithms have been developed to address the computational challenges of the generalized maximum flow problem (GMFP). Among the foundational works, Ahuja, Magnanti, and Orlin [1] provided a comprehensive treatment of flow problems, including the generalized case. Their influential book detailed a range of sequential algorithms and introduced advanced techniques, such as scaling and cost adjustments. These approaches became instrumental in improving the efficiency and accuracy of generalized flow computations, laying the groundwork for future developments.

In 2016, Végh [4] advanced the field by presenting polynomial-time algorithms specifically designed for generalized flow problems using strongly polynomial methods. This work addressed critical inefficiencies in earlier approaches, offering robust performance

guarantees and significant computational improvements. Végh's algorithms are particularly notable for their theoretical contributions to the efficiency of sequential computation in the GMFP, bridging the gap between practical application and mathematical rigor.

Recognizing the limitations of sequential methods in large-scale scenarios, Cherkassky and Goldberg [9] adapted the push-relabel algorithm to handle generalized flows, with a focus on parallel scalability. Their adaptation demonstrated how the careful handling of gain factors—a distinguishing characteristic of generalized flows—could ensure both accuracy and convergence. By leveraging the inherent parallelizability of the push-relabel method, they highlighted its potential to scale across modern multi-core and distributed architectures. Their contributions have since inspired numerous parallelization efforts, advancing the applicability of GMFP solutions to increasingly complex and large-scale network problems.

The paper by Goldberg, Plotkin, and Tardos [10] presents a significant contribution to the field of network optimization by addressing the generalized circulation problem (GCP), an extension of the classical circulation problem. The authors propose combinatorial algorithms to solve the GCP efficiently, focusing on their theoretical complexity and practical implementability. By leveraging combinatorial techniques, they aim to avoid reliance on more computationally intensive linear programming methods. They propose two weakly polynomial algorithms to handle GCP. In the first algorithm, at each iteration, the algorithm calculates a minimum-cost pseudoflow using the cost, denoted by $\mathrm{c}(\mathrm{e}) = -log(\alpha(e))$, attached to each arc of the network. Based on this pseudoflow, the solution is updated at each iteration. The second algorithm saturates all flow-generating cycles, resulting in excesses at some nodes of the network. Then, the algorithm iteratively finds a path from an excess node to the source, and the flow is increased on this path until no such path exists.

In this paper, without any loss of generality, it is assumed that the gain/loss function is symmetric, meaning $\alpha(u, v) = 1/\alpha(v, u)$. If this condition is not met, a symmetric arc can be introduced with zero capacity to enforce this property. Also, the maximum flow problem in a network with gains/losses follows constraints similar to the traditional maximum flow problem.

The generalized maximum flow problem consists in maximizing the flow between two given nodes (source and, respectively, sink), considering both the capacity constraints of the arcs and the fact that each arc may either loss or gain flow. A specific example can be seen in the network illustrated in Figure 1, where S is the source node and T is the sink node. If 10 units of flow are sent through arc $(S,\ A)$, which has a gain/loss factor $\alpha(S, A) = 0.9$, only 9 units will reach node A.

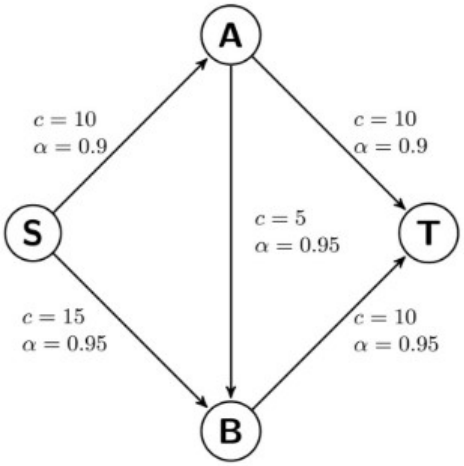

Figure 1. Example of a network with losses.

To determine the residual network, the gain/loss factor for each arc must be taken into account. Given a flow $f(u,\ v)$ on arc (u,v), the residual capacity function is

$$\begin{cases} c_f\ (u,v) = \min\{c(u,v), c(u,v)/\alpha(u,v)\} - f(u,\ v) \\ c_f\ (v,u) = f(u,\ v)/\alpha(u,v) \end{cases}$$

2. Adaptation of the Ford–Fulkerson Algorithm for Solving the Maximum Flow Problem in a Generalized Network

A generalized augmenting path (GAP) $H = (u_1, u_2, \ldots, u_h = u_1, v_{h+1}, \ldots, v_k = t)$ ($h \geq 2$ and $k \geq h$), so that $C = (u_1, u_2, \ldots, u_h = u_1)$ is a flow generating cycle in G_f, i.e., $\alpha(u_1, u_2) \cdot \alpha(u_2, u_3) \cdot \ldots \cdot \alpha(u_{h-1}, u_h) > 1$ and $P = (u_h, v_{h+1}, \ldots, v_k = t)$, is a directed path in G_f that connects the cycle C with the sink node t. We also denote $H = C \bigcup P$. If $k = h$, then the path P is empty and the node t belongs to C.

The following theorem gives an optimality condition for the GMFP.

Theorem 1. *A generalized flow f is a generalized maximum flow in G if and only if there is no $s - t$ path and no GAP in the residual network G_f* [11].

Using Theorem 1, Algorithm 1 is obtained to solve the GMFP. In Algorithm 1, while there is a directed $s - t$ path or a GAP in G_f, the flow is increased on such a path or GAP. To do this, Algorithms 2 and 3 are used. The search of an $s - t$ path in G_f can be done using the Minimum Loss Path Algorithm (AMLP) [12] for determining the path with minimal loss. Since the AMLP is designed to find a path in a network that minimizes the losses from the source node to the sink node, the use of the AMLP helps Algorithm 1 to increase the flow faster. Moreover, since the AMLP is based on the Bellman–Ford algorithm (BFA), the AMLP can be easily adapted to find a GAP as well by applying BFA backwards (from sink t to source s). Therefore, when applying the BFA in this manner, an MLP or a GAP is found.

Algorithm 1. Adaptation of the Ford–Fulkerson algorithm for solving the maximum flow problem in a generalized network

Input:

- A network $G = (V,\ E,\ s,\ t,\ c,\ \alpha)$

Output:

- f—the flow with minimal losses through G

Initialize a feasible flow $f = 0$
Initialize the residual network $G_f = G$
foreach $(u,\ v)\ \in\ E$ **do**
  $\alpha_f\ (u,\ v)\ \leftarrow\ \alpha(u,\ v)$
  $\alpha_f\ (v,\ u)\ \leftarrow\ 1/\alpha(u,v)$
end foreach;
while exists a GAP or an augmenting path from s to t in G_f **do**
  If a GAP H is found **then**
    Update the residual network G_f on H using Algorithm 3
  else
    Find an augmenting path P in G_f using Algorithm MLP [12]
    Update the residual network G_f on P using Algorithm 2
  end if;
end while;

Algorithm 2. Algorithm for updating the residual network on an $s - t$ path

Input:

- Residual network $G_f = \left(V,\ E,\ s,\ t,\ c_f,\ \alpha_f\right)$
- Augmenting path P in G_f from x to y

Output:

- Updated residual network G_f

//Determine the flow value at node t on path P
Consider a flow, $f = c_f\ (s,\ u)$, where $(s,\ u)\ \in\ P$ is the first arc in P
$u\ \leftarrow\ s$
while $u\ \neq\ t$ **do**
  Consider the arc $(u,\ v)\ \in\ P$
  $f\ \leftarrow\ min\left(f\ , c_f\ (u,\ v)\right) \cdot\ \alpha_f\ (u,\ v)$
  u ← v
end while;
//Update G_f
$v\ \leftarrow\ t$
while $v\ \neq\ s$ **do**
  Consider the arc $(u,\ v)\ \in\ P$
  $c_f\ (u,\ v) \leftarrow c_f\ (u,\ v) - \frac{f}{\alpha_f (u,\ v)}$
  $c_f\ (v,\ u)\ \leftarrow\ c_f\ (v,\ u) + f$
  $f\ \leftarrow\ \frac{f}{\alpha_f\ (u,\ v)}$
  $v\ \leftarrow\ u$
end while;

Algorithm 3. Algorithm for updating the residual network on a GAP

Input:

- Residual network $G_f = \left(V,\ E,\ s,\ t,\ c_f,\ \alpha_f\right)$
- a GAP $H = C \bigcup P$ in G_f

Output:

- Updated residual network G_f

if P is empty **then**
  $x = t$
else
  x is the exit node from the cycle C of the gap H towards node t
end if;
//Determine the flow value at node x on the cycle C
Consider $f = c_f\ (x,\ u)$, where $(x,\ u)\ \in\ C$
$u\ \leftarrow\ x$
//starting from node $u = x$*, iterate the arcs of* C
while $u \neq t$ **do**
  Consider the arc $(u,\ v)\ \in\ H$
  $f\ \leftarrow\ min\left(f\ , c_f\ (u,\ v)\right) \cdot\ \alpha_f\ (u,\ v)$
  u ← v**end while;**
//Update G_f
$v\ \leftarrow\ t$
$count\ \leftarrow\ 0$

Algorithm 3. *Cont.*

```
//the update of G_f is done backwards on H starting from node t
while count < 2 do //the while loop stops when node x is found second time
  Consider the arc (u, v) ∈ H
  c_f (u, v) ← c_f (u, v) − f / α_f(u, v)
  c_f (v, u) ← c_f (v, u) + f
  f ← f / α_f (u, v)
  v ← u
  if v = x then
    count ← count + 1
  end if;
end while;
```

Each iteration of the proposed algorithm operates in two phases. In the first phase, an augmenting residual s-t path P or a GAP is searched. Then, the value of the flow f on this path/GAP is set to the residual capacity of the first arc. The arcs of the path/GAP are then iterated, and the flow is updated by selecting the minimum value between the current value of f and the residual capacity $c_f(u, v)$, taking into account the gain/loss factor. In the second phase, the residual network G_f is updated backwards, starting from the sink node t. During this process, on each arc (v, u) of the path/GAP, the value f is adjusted according to the gain/loss factor $\alpha(v, u)$ together with the residual capacity $c_f(u, v)$. The algorithm ends when no path or GAP is found anymore.

Theorem 2 (correctness of the algorithm)**.** *If the capacities and gain/loss factors are all rational values, Algorithm 1 finds the generalized maximum in a finite number of iterations.*

Proof. At each iteration of Algorithm 1, the value of the flow increases on a residual directed s-t path or on a GAP. Since the values of the capacities and gain/loss factors are rational numbers, the value of the generalized maximum flow is reached in a finite number of iterations. Using Theorem 1, it follows that the flow when Algorithm 1 ends is optimum. □

3. Example

To illustrate the proposed algorithm, an example is provided in Figure 2. At the start, the residual network is G_f, with the source node designated as $s = 1$ and the sink node as $t = 6$. We iteratively increase the flow on minimal loss paths from source node 1 to sink node 6 and increase the flow on GAPs in the residual network G_f.

Iteration 1

In the first iteration, the minimal loss path found in G_f is the path $P = 1 \to 2 \to 3 \to 6$ (the loss factor is $\alpha_f(1,\ 2) \cdot \alpha_f(2,\ 3) \cdot \alpha_f(3,\ 6) = 0.5 \cdot 0.75 \cdot 0.7 = 0.2625$). On this path, the initial flow is $f = c_f(1,\ 2) = 40$. On the arc $(1,\ 2)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(1,\ 2) = 20, c_f(1,\ 2) \cdot \alpha_f(1,\ 2) = 20\Big\}$$

Thus, 20 units of flow reach node A. On the arc $(2,\ 3)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(2,\ 3) = 15, c_f(2,\ 3) \cdot \alpha_f(2,\ 3) = 33.75\Big\}$$

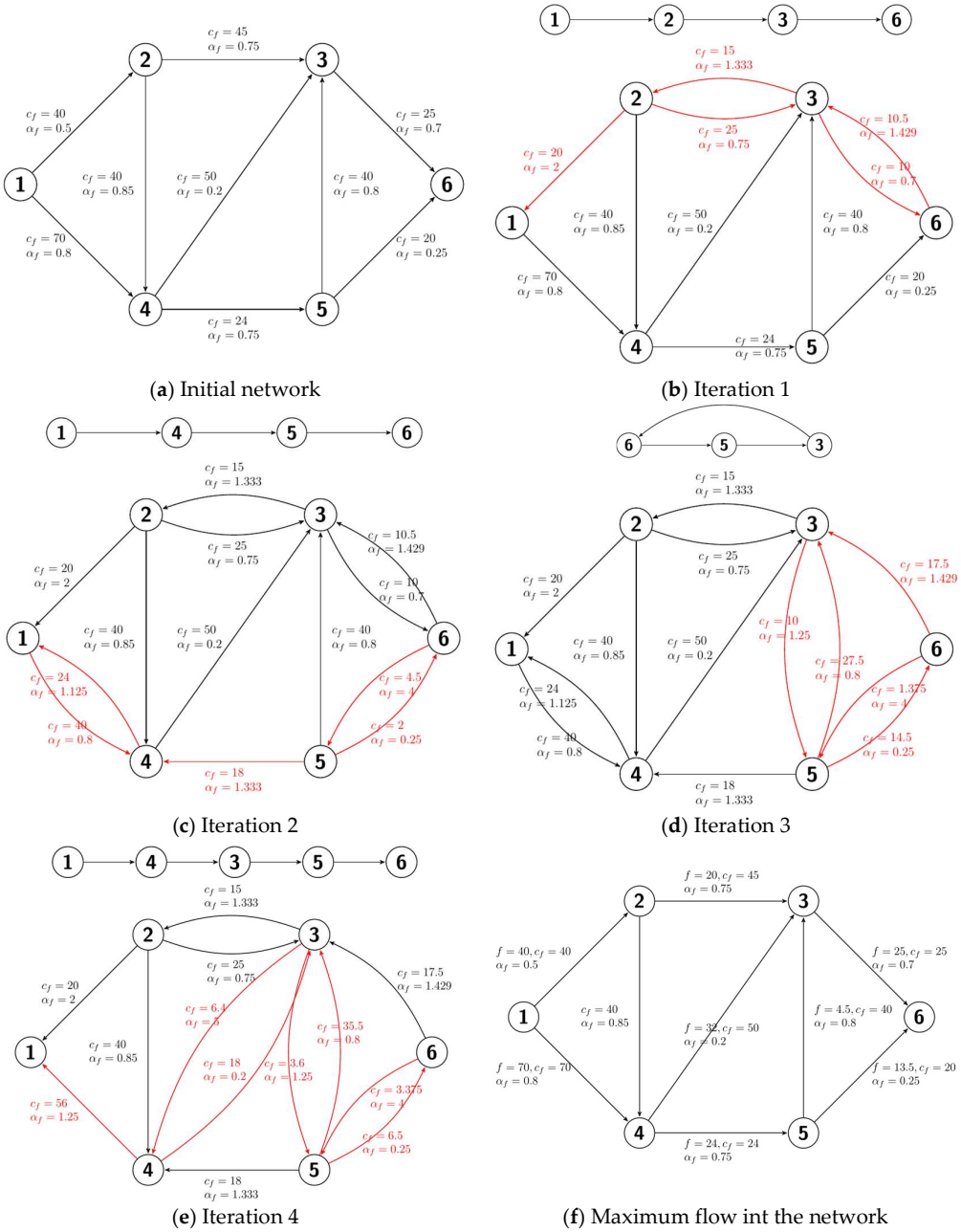

Figure 2. Example illustrating the determination of maximum flow in a network with losses. In order to highlight the progress of the algorithm, the s-t paths and cycle found by the algorithm are presented in red at each iteration.

Therefore, 15 units of flow reach node 3. From node 3 to sink 6, the flow is $f = min\left\{f \cdot \alpha_f(3,\ 6) = 10.5, c_f(3,\ 6) \cdot \alpha_f(3,\ 6) = 17.5\right\}$. Thus, 10.5 units reach the sink.

According to Algorithm 2, the updated residual network is determined by starting with the flow f on the minimal loss path found, in reverse, from the sink node towards the source node. Thus, starting from 6, the arc $(6,\ 3)$ will have the capacity

$$c_f(6,\ 3) = c_f(6,\ 3) + f = 10.5 \text{ and } c_f(3,\ 6) = 10.$$

The gain/loss factor $\alpha_f(6,\ 3) = \frac{1}{0.7} = 1.(428,571)$, so 15 units of flow enter node 3:

$$f \cdot \alpha_f(6,\ 3) = 15.$$

For the arc $(2,\ 3)$, we have

$$c_f(2,\ 3) = c_f(2,\ 3) - \frac{15}{0.75} = 25, c_f(3,\ 2) = 15, \text{with } \alpha_f(3,\ 2) = \frac{1}{0.75} = 1.(3).$$

For the arc $(1,\ 2)$, we have

$$c_f(1,\ 2) = c_f(1,\ 2) - \frac{20}{0.5} = 0, c_f(2,\ 1) = 20, \text{with } \alpha_f(2,\ 1) = \frac{1}{0.5} = 2.$$

Iteration 2

In the second iteration, the minimal loss path found in G_f is $P = 1 \to 4 \to 5 \to 6$ (the loss factor is $\alpha_f(1,\ 4) \cdot \alpha_f(4,\ 5) \cdot \alpha_f(5,\ 6) = 0.8 \cdot 0.75 \cdot 0.25 = 0.15$). On this path, the initial flow is $f = c_f(1,\ 4) = 60$. On the arc $(1,\ 4)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(1,\ 4) = 48, c_f(1,\ 4) \cdot \alpha_f(1,\ 4) = 48\Big\}$$

Thus, 48 units of flow reach node 4. On the arc $(4,\ 5)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(4,\ 5) = 36, c_f(4,\ 5) \cdot \alpha_f(4,\ 5) = 18\Big\}$$

Therefore, 18 units of flow reach node 5. Next, on the arc $(5,\ 6)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(5,\ 6) = 4.5, c_f(5,\ 6) \cdot \alpha_f(5,\ 6) = 5\Big\}$$

The residual network is updated starting with the flow $f = 4.5$. Thus, starting from 6, the arc $(6,\ 5)$ will have the capacity

$$c_f(6,\ 5) = c_f(6,\ 5) + f = 4.5 \text{and } c_f(5,\ 6) = 2$$

The gain/loss factor $\alpha_f(6,\ 5) = \frac{1}{0.25} = 4$, so 18 units of flow enter node 5 $f \cdot \alpha_f(6,\ 5) = 18$. For the arc $(4,\ 5)$, we have

$$c_f(4,\ 5) = c_f(4,\ 5) - \frac{18}{0.75} = 0 \text{ and } c_f(5,\ 4) = 18 \text{ with } \alpha_f(5,\ 4) = \frac{1}{0.75} = 1.(3).$$

For the arc $(1,\ 4)$, we have

$$c_f(1,\ 4) = c_f(1,\ 4) - \frac{24}{0.8} = 30 \text{ and } c_f(4,\ 1) = 24 \text{ with } \alpha_f(5,\ 4) = \frac{1}{0.8} = 1.25.$$

Iteration 3

In the third iteration, a GAP $H = 6 \to 5 \to 3 \to 6$ is found. Since the node t is part of the cycle $C = H$, we have $x = t = 6$. The gain factor on H is $\alpha_f(6,\ 5) \cdot \alpha_f(5,\ 3) \cdot \alpha_f(3,\ 6) = 4 \cdot 0.8 \cdot 0.7 = 2.24$. The initial flow is $f = c_f(6,\ 5) = 4.5$. On the arc (4.6), the flow f is $f = \min\Big\{f \cdot \alpha_f(6,\ 5) = 18,\ c_f(6,\ 5) \cdot \alpha_f(6,\ 5) = 18\Big\}$.

Thus, 18 units of flow reach node 5. On the arc $(5,\ 3)$, the flow f is

$$f = \min\Big\{f \cdot \alpha_f(5,\ 3) = 14.4, c_f(5,\ 3) \cdot \alpha_f(5,\ 3) = 14.4\Big\}$$

Therefore, 18 units of flow reach node 5. Next, on the arc $(3,\ 6)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(3,\ 6) = 9.8, c_f(3,\ 6) \cdot \alpha_f(3,\ 6) = 7\Big\}$$

The residual network is updated starting with the flow $f = 7$. Thus, starting from 6, the arc $(6,\ 3)$ will have the capacity

$$c_f(6,\ 3) = c_f(6,\ 3) + f = 17.5 \text{and } c_f(3,\ 6) = 0$$

In total, 10 units of flow enter node 3 : $f \cdot \alpha_f(6,\ 3) = 10$. For the arc $(5,\ 3)$, we have

$$c_f(5,\ 3) = c_f(5,\ 3) - \frac{10}{8} = 27.5 \text{ and } c_f(3,\ 5) = 10.$$

Furthermore, 12.5 units of flow enter node 5. For the arc $(5,\ 6)$, we have

$$c_f(6,\ 5) = c_f(6,\ 5) - \frac{12.5}{0.25} = 1.325 \text{ and } c_f(5,\ 6) = 14.5.$$

So, the GAP H was eliminated.

Iteration 4

In the fourth iteration, the minimal loss path found in G_f is $P = 1 \to 4 \to 3 \to 6$ (the loss factor is $\alpha_f(1,\ 4) \cdot \alpha_f(4,\ 3) \cdot \alpha_f(4,\ 5) \cdot \alpha_f(5,\ 6) = 0.8 \cdot 0.2 \cdot 1.25 \cdot 0.25 = 0.05$). On this path, the initial flow is $f = c_f(1,\ 4) = 40$. On the arc $(1,\ 4)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(1,\ 4) = 32, c_f(1,\ 4) \cdot \alpha_f(1,\ 4) = 32\Big\}$$

Thus, 32 units of flow reach node 4. On the arc $(4,\ 3)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(4,\ 3) = 6.4, c_f(4,3) \cdot \alpha_f(4,3) = 10\Big\}$$

Thus, 2 units of flow reach node 3. On the arc $(3,\ 5)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(3,\ 5) = 8, c_f(3,\ 5) \cdot \alpha_f(3,\ 5) = 12.5\Big\}$$

Therefore, 8 units of flow reach node 5. On the arc $(5,\ 6)$, the flow f is

$$f = min\Big\{f \cdot \alpha_f(5,\ 6) = 2, c_f(5,\ 6) \cdot \alpha_f(5,\ 6) = 3.625\Big\}$$

Therefore, 8 units of flow reach node 2.

The residual network is updated starting with the flow $f = 2$. Thus, starting from 6, the arc $(6,\ 5)$ will have the capacity

$$c_f(6,\ 5) = c_f(6,\ 5) + f = 3.375 \text{ and } c_f(5,6) = 6.5$$

In total, 8 units of flow reach node 5: $f \cdot \alpha_f(6,\ 5) = 8$.

For the arc $(3,\ 5)$, we have

$$c_f(3,\ 5) = c_f(3,\ 5) - \frac{8}{1.25} = 3.6 \text{ and } c_f(5,\ 3) = 35.5.$$

For the arc $(4,\ 3)$, we have

$$c_f(4,\ 3) = c_f(4,\ 3) - \frac{6.4}{0.2} = 18 \text{ and } c_f(3,\ 4) = 6.4$$

For the arc $(1,\ 4)$, we have

$$c_f(4,\ 3) = c_f(4,\ 3) - \frac{32}{1.25} = 0 \quad (1)$$

The maximum flow in the example presented is 20.875. Since in the residual network from Figure 2d there is no path from source node 1 to sink node 6 and there is no GAP, according to Theorem 2, it follows that the corresponding flow in the network G from Figure 2e is optimum.

4. Results and Discussion

For the implementation of the proposed algorithm, both sequential and parallel versions of the GPU were utilized. The GPU implementation of the algorithm consists in applying the parallel Bellman–Ford algorithm for determination of each minimum loss path. The algorithm starts with the distance for the source set to 0 and the distances from the source to the rest of the nodes set to infinity. The algorithm sequentially relaxes the arcs of the graph where a shorter path from source to a node of the graph is found. If such a path is determined, the distance label of the node is modified. Since any shortest path in the graph can have no more than $n - 1$ edges (the graph has "n" nodes in total), the algorithm has $n - 1$ relaxation iterations. After the last iteration, if there still is a node that can be relaxed, this means that a negative cycle (of which the sum of weights of the arcs is negative) exists in the graph. If a negative cycle is found, then there is no shortest path from the source and any node from the cycle. To implement the Bellman–Ford algorithm on GPUs, kernels are used that are executed on separate threads. Using the calculated shortest path, for each node of the path, the minimum distance for the successor node is calculated. When a new shortest path is calculated for a node, its distance is modified, and so, a new vector of distances is created, which replaces the previous vector. The execution of the Bellman–Ford algorithm ends when the new vector of distances is equal to the previously calculated vector of distances. The pseudocode of the kernel is presented in [12].

As shown in Algorithm 1, its overall complexity is driven by the complexity of the MLP algorithm, multiplied by the number of iterations needed to find an augmenting path. Table 1 summarizes the experimental results obtained from sequential CPU implementation and parallel GPU implementation. For the tests from this paper, a laptop with Intel(R) Core(TM) i7-10750H CPU @ 2.60 GHz 2.59 and with NVIDIA GeForce RTX 2060 GPU system was used. The operating system of the laptop is Windows 10.

The GPU efficiency initially increases with the complexity of the problem, but after a certain point, it begins to decrease, possibly due to GPU resource saturation. The speed-up ranges between 1.04 and 5.72, with maximum values occurring for medium-sized problems. GPU performance is significantly superior to CPU for most tested configurations, especially for medium and large graph sizes.

As shown in Figure 3, the execution time on the GPU is significantly shorter than on the CPU for all the dense networks analyzed, regardless of the number of nodes. The efficiency of the GPU is especially evident in large, dense networks. Thus, the execution time on the GPU increases more slowly compared to the CPU as the number of nodes grows. The speed-up ratio decreases as the number of nodes increases. For smaller node counts (2000–10,000), the GPU provides a considerable speed-up (2.7–5.7 times). For larger node counts (15,000–25,000), this ratio decreases slightly but remains significant (2.98–4.45 times).

As the number of arcs increases from 37 M to 135 M, the execution time on both the CPU and GPU increases. However, the rate of increase is more pronounced for the CPU, suggesting that the complexity of the problem disproportionately affects CPU performance

compared to the GPU. For example, the CPU execution time at 135 M arcs is the highest (Figure 4), indicating that handling larger datasets significantly burdens CPU resources.

Table 1. Execution time and speed-up for Algorithm 1.

Number of Nodes	Number of Arcs	CPU Execution Time (s)	GPU Execution Time (s)	Speed-Up
2000	266,863	0.08897	0.085364	1.042243
	545,296	0.2993	0.269048	1.112441
	792,044	0.72046	0.585024	1.231505
	1,024,298	1.25296	0.95729	1.308861
5000	1,730,230	2.26628	0.8387	2.702134
	3,260,184	6.1682	2.075064	2.972535
	4,667,223	9.50437	2.87175	3.309609
	6,193,292	12.7624	4.712715	2.708078
10,000	6,780,947	26.096	5.43481	4.80164
	13,011,798	58.9848	12.6801	4.651762
	18,582,466	90.47255	18.02492	5.019304
	24,094,464	140.5387	24.57208	5.719448
15,000	14,574,815	78.0297	15.6064	4.999853
	28,771,632	153.2161	37.87079	4.045761
	41,488,314	248.9598	55.9062	4.45317
	54,199,030	345.1695	78.51689	4.894848526
20,000	24,619,509	155.05946	38.430896	4.03476047
	47,347,736	334.5968	84.30291	3.968982803
	68,784,930	500.3322	135.3364	3.696952187
	89,281,904	632.392	179.08824	3.53117547
25,000	36,956,626	386.6308	103.686	3.728864
	71,570,683	767.352	223.0736	3.439905
	105,252,771	1274.99	356.5916	3.57549
	135,731,348	1516.948	509.3967	2.977931

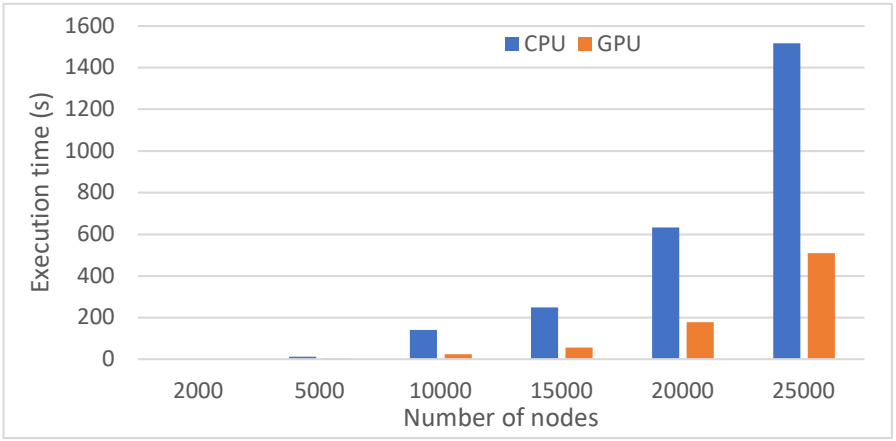

Figure 3. Execution times for Algorithm 1 in dense networks.

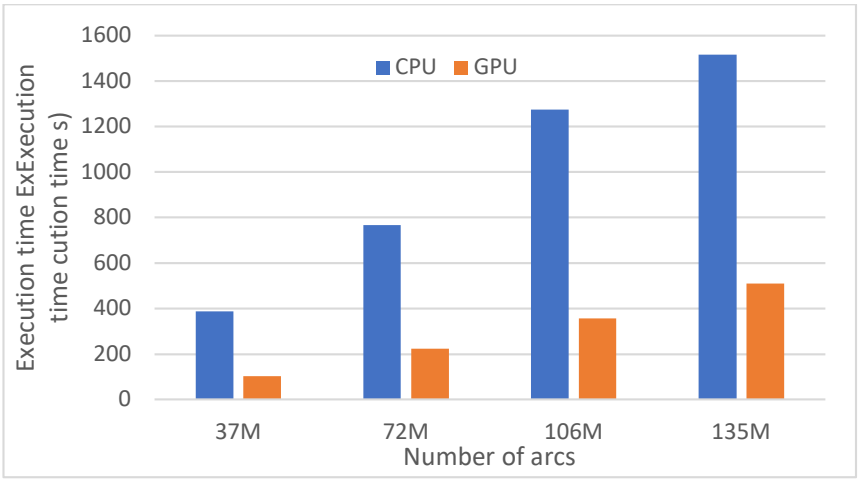

Figure 4. Execution times for Algorithm 1 for different network densities.

5. Conclusions

The proposed adaptation of the Ford–Fulkerson algorithm for maximum flow in generalized networks addresses the complexities introduced by gain/loss factors in arcs. This new approach allows for the efficient determination of flow in networks where flow can be lost or gained, making it applicable to a wider range of real-world scenarios, such as transportation and communication networks. The GPU implementation shows a clear advantage over the CPU, particularly in dense networks and for medium to large graph sizes. The execution time on the GPU is consistently shorter across all configurations, demonstrating the superior efficiency of parallel processing. The GPU performs particularly well for dense networks and medium to large graph sizes. In these cases, the parallelism of the GPU can be fully leveraged, yielding notable performance gains over sequential CPU implementations. The proposed solution enhances the algorithm's capability to effectively manage flow with losses or gains, further expanding its applicability.

Author Contributions: Conceptualization, A.M.D., D.E.S. and J.T.; methodology, D.E.S.; software, D.E.S. and A.M.D.; validation, A.M.D., D.E.S. and J.T.; formal analysis, J.T.; investigation, A.M.D.; resources, D.E.S.; data curation, D.E.S.; writing—original draft preparation, A.M.D., D.E.S. and J.T.; writing—review and editing, A.M.D., D.E.S. and J.T.; visualization, A.M.D.; supervision, A.M.D. and J.T.; project administration, D.E.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: Authors declare no conflict of interests.

References

1. Ahuja, R.; Magnanti, T.; Orlin, J. *Network Flows: Theory, Algorithms, and Applications*; Prentice Hall: Englewood Cliffs, NJ, USA, 1993.
2. Jewell, W. Optimal flow through networks with gains. *Oper. Res.* **1962**, *10*, 476–499. [CrossRef]
3. Tardos, E.; Wayne, K. Simple generalized maximum flow. Integer Programming and Combinatorial Optimization. In Proceedings of the 6th International IPCO Conference, Houston, TX, USA, 22–24 June 1998; pp. 310–314.
4. Vegh, L. A strongly polynomial algorithm for generalized flow maximization. In Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 19–21 June 2016; pp. 644–653.
5. Tayyebi, J.; Deaconu, A. Inverse generalized maximum flow problems. *Mathematics* **2019**, *8*, 899. [CrossRef]

6. Spridon, D.E.; Deaconu, A.M.; Tayyebi, J. New approach for the Generalized Maximum Flow Problem. In Proceedings of the International Conferences on Applied Computing, Zagreb, Croatia, 26–28 October 2024; pp. 29–36.
7. Dantzig, G.B. Application of the simplex method to a transportation problem. In *Activity Analysis of Production and Allocation*; Koopmans, T.C., Ed.; John Wiley and Sons: New York, NY, USA, 1951; pp. 359–373.
8. Ford, L.R., Jr.; Fulkerson, D.R. Maximal flow through a network. *Can. J. Math.* **1956**, *8*, 399–404. [CrossRef]
9. Cherkassky, B.; Goldberg, A. On Implementing the Push—Relabel Method for the Maximum Flow Problem. *Algorithmica* **1997**, *19*, 390–410. [CrossRef]
10. Goldberg, A.V.; Plotkin, S.A.; Tardos, É. Combinatorial algorithms for the generalized circulation problem. *Math. Oper. Res.* **1991**, *16*, 351–381. [CrossRef]
11. Wayne, K. Generalized Maximum Flow Algorithms. 1999. Available online: https://www.cs.princeton.edu/~wayne/papers/thesis.pdf (accessed on 5 December 2024).
12. Deaconu, A.; Spridon, D.; Ciupala, L. 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.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.