A NEW POLYNOMIAL ALGORITHM FOR THE CONSTRAINED MAXIMUM FLOW PROBLEM

Cenk Çalışkan
Department of Finance and Economics
Woodbury School of Business
Utah Valley University
Orem, UT 84058
phone: (801) 863-6487
fax: (801) 863-7218
e-mail: cenk.caliskan@uvu.edu

ABSTRACT

The constrained maximum flow problem is a variant of the classical maximum flow problem in which the total cost of the flow from the source to sink is constrained by a budget limit. It is important to study this problem because it has many important practical applications. In this research, we present a new polynomial time algorithm that is based on the cost scaling algorithm for the minimum cost network flow problem. We prove that it runs in $O(n^2 m \log(nC))$ worst case time.

Keywords: network flows, maximum flow, minimum cost network flow, scaling, complexity analysis

1 Introduction

The constrained maximum flow problem is a variant of the maximum flow problem. The maximum flow problem has many direct applications, as well as applications where it appears as a subproblem within the context of larger problems in transportation, logistics, telecommunications, and many other fields. Many algorithms were proposed to solve the maximum flow problem to date. Ahuja et al. (1993) describes a number of applications for the classical maximum flow problem, as well as many of the important algorithmic developments on the problem. On the other hand, very few articles appeared in the literature for variants of the classical maximum flow problem. Ahuja and Orlin (1995) propose an $O(mS(n, m, nC) \log U)$ capacity scaling algorithm, where n is the number of nodes in the network; m, the number

of arcs; C, the largest arc cost; U, the largest arc capacity; and $S(n, m, nC)$, the time to find a shortest path from a single source to all other nodes where nonnegative reduced costs are used as arc lengths. Çalışkan (2008) proposes a double scaling algorithm that runs in $O(nm \log(n^2/m) \log m \log U \log(nC))$ time with the use of *dynamic trees* due to Goldberg and Tarjan (1990). Our algorithm is based on the cost scaling algorithm for the minimum cost flow problem due to Goldberg and Tarjan (1987), which in turn, is the improved version of the algorithm in Röck (1980).

2 Problem Description

Let $G = (N, A)$ be a directed network consisting of a set N of nodes and a set A of arcs. In this network, the flow on each arc (i, j) is represented by the nonnegative variable x_{ij} with a cost c_{ij} and capacity u_{ij}. The constrained maximum flow problem is to send the maximum possible flow from a source node $s \in N$ to a sink node $t \in N$ where the total cost of the flow is constrained by the budget, D. For simplicity of exposition, we assume that there is an arc $(t, s) \in A$ with $c_{ts} = 0$ and $u_{ts} = \min\{\sum_{(s,i)\in A} u_{si}, \sum_{(i,t)\in A} u_{it}\}$, an upper bound on the maximum flow from s to t. The problem then can be formulated as follows:

$$\max \quad x_{ts} \tag{1}$$

$$s.t.$$

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0 \quad \forall i \in N \tag{2}$$

$$\sum_{(i,j)\in A} c_{ij} x_{ij} \leq D \tag{3}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in A \tag{4}$$

We assume that all arc capacities and all arc costs are nonnegative integers, and there exists a directed path between every pair of nodes in the network. These assumptions are not restrictive because fractional capacities and costs can be made integers by multiplying by a suitably large number, and nonnegativity and connectivity assumptions can be satisfied by applying the transformations mentioned in Ahuja and Orlin (1995).

3 Motivation

The constrained maximum flow problem has many potential applications. We provide a few such applications here. Consider an agency that would like to allocate its funds on a number of activities with associated costs. The objective is to maximize the number of selected activities subject to a budget. Each activity has an associated start and end time and some activities may require selecting from a number of subsequent activities. We can formulate this as the constrained integer maximum flow problem as follows: (i) create a node for each activity; (ii) connect node s to each main activity node with a directed unit capacity arc with cost equal to the activity cost; (iii) connect each main activity node to its alternate subsequent activity nodes with arcs of unit capacity and cost equal to the cost

of each subsequent activity, which will form a tree rooted at each main activity node with all arcs directed away from the root; (iv) connect each leaf activity node to node t with a directed unit arc of zero cost.

Another example is as follows: Consider an agency that would like to allocate its funds to train its employees. The objective is to maximize the number of employees trained. There are a number of courses with associated costs and class size limits, as well as start and end times. Selecting a course may require a subsequent one from a number of alternatives. This can also be formulated similar to the problem explained above; except that capacities can be greater than 1.

Assume that we would like to determine the capacity of a distribution network. Some distribution centers are origins, and others are destinations for the flow of goods. Each arc represents a mode of transportation between two distribution centers and has an associated cost and capacity. The objective is to determine the maximum flow between the origin and destination nodes subject to the transportation budget. This problem can be formulated as the constrained maximum flow problem by introducing a super source and a super sink node and connecting the super source to the origins and the destinations to the super sink by directed arcs of unlimited capacity and zero cost.

4 Preliminaries

We denote the largest arc capacity in G by U, the largest arc cost by C, the number of nodes by n, and the number of arcs by m. For each node $i \in N$, we define $F(i, G) = \{j \in N | (i, j) \in G\}$ and call it the *forward star* of i in G. Similarly, for each node $i \in N$, we define $B(i, G) = \{j \in N | (j, i) \in G\}$ and call it the *backward star* of i in G. To represent a path or a partial path, we use predecessor indices. The index of the node preceding a given node i in the current path is denoted by $pred(i)$.

A *flow* is a function $x : A \rightarrow R^+ \cup \{0\}$ that satisfies Eqs. 2 and 4. The value of x_{ts} is called the value of flow x. If a flow x also satisfies the budget constraint (Eq. 3), then we call it a *feasible flow*. The constrained maximum flow problem is to find a feasible flow of maximum value. A *pseudoflow* is a function $x : A \rightarrow R^+ \cup \{0\}$ that satisfies only Eq. 4. For a pseudoflow x, we define the imbalance of node $i \in N$ as follows:

$$e(i) = \sum_{(j,i) \in A} x_{ji} - \sum_{(i,j) \in A} x_{ij} \qquad \text{for all } i \in N. \tag{5}$$

If $e(i) > 0$ for some node i, then we call node i an *excess* node, and we call $e(i)$, the *excess* of node i. We also call a node that is an *excess* node, an *active* node. If $e(i) < 0$ for some node i, then we call node i a *deficit* node, and we call $-e(i)$, the *deficit* of node i. If $e(i) = 0$ for some node i, then we call node i a *balanced* node. We also call a *deficit* or a *balanced* node, an *inactive* node. A pseudoflow with $e(i) = 0$ for all $i \in N$ is a flow.

Given a flow or pseudoflow x, the corresponding *residual network* $G(x)$ is defined as follows: We replace each arc $(i,j) \in G$ with two arcs: (i,j) and its reversal (j,i), where $c_{ji} = -c_{ij}$. The residual capacity of (i,j) is defined as $r_{ij} = u_{ij} - x_{ij}$, and the residual capacity of (j,i), as $r_{ji} = x_{ij}$. We only include in $G(x)$ arcs with positive residual capacity. We associate a node potential $\pi(i)$ for each node $i \in N$. With respect to a set of node potentials π, we define the reduced cost of an arc (i,j) as $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$. We use the concept of ϵ-optimality of the minimum cost flow problem in our cost scaling algorithm. This concept is due to Bertsekas (1979), and independently, Tardos (1985).

Definition 1 (**Epsilon Optimality**). *A flow x or a pseudoflow x is called ϵ-optimal for some ϵ and π, if $c_{ij}^{\pi} \geq -\epsilon$ for all (i,j) in the residual network $G(x)$.*

It is well-known that an ϵ-optimal flow is optimal for the minimum cost flow problem if $\epsilon = 0$. When the arc costs are integers, any ϵ-optimal flow is optimal for $\epsilon < \frac{1}{n}$, as the following lemma establishes:

Lemma 1. *For a minimum cost network flow problem with integer arc costs, if $\epsilon \geq C$, any flow x is ϵ-optimal. If $\epsilon < 1/n$, then any ϵ-optimal flow is an optimal flow.*

Proof: See Ahuja et al. (1993), page 363.
Our cost scaling algorithm also uses the following theorem from Ahuja and Orlin (1995):

Theorem 1 (Ahuja and Orlin (1995)). *Let x^* be an optimal solution to the minimum cost flow problem with the same cost vector as the constrained maximum flow problem and with a source supply equal to the optimal flow value for the constrained maximum flow problem. Then, x^* is also an optimal solution to the constrained maximum flow problem if $cx^* = D$.*

Proof: See Ahuja and Orlin (1995), page 91.
Based on Theorem 1 and Lemma 1, the constrained maximum flow problem can be solved by modifying the cost scaling algorithm for the minimum cost flow problem. In such a procedure, one will first obtain an initial flow x for which $cx = D$ and the flow is ϵ-optimal with $\epsilon = C$; and gradually convert this flow to a flow that satisfies ϵ-optimality with smaller ϵ, increasing the flow value, until ϵ is small enough, i.e. $\epsilon < \frac{1}{n}$, so that the optimal solution to the constrained maximum flow problem is obtained. Our cost scaling algorithm is based on this idea.

We call an ϵ-optimal flow x that also satisfies the condition $cx = D$ an *ϵ-optimal solution* (to the constrained maximum flow problem). We call an arc (i,j) in the residual network $G(x)$ an *admissible arc* if $-\frac{\epsilon}{2} \leq c_{ij}^{\pi} < 0$, and a path consisting entirely of admissible arcs, an *admissible path*. We define the *admissible network* of a given residual network as the network consisting solely of admissible arcs. We represent the outgoing arcs of a node i, i.e. the forward star of i, with $F(i)$, and the incoming arcs of node i, i.e. backward star of i, with $B(i)$ (in the residual network $G(x)$, with respect to a flow or pseudoflow x). We also represent the admissible paths with predecessor indices. For a given admissible path, $pred(i)$ represents the node on the path that comes immediately before node i.

5 The Cost Scaling Algorithm

In this section, we present a cost scaling algorithm to solve the constrained maximum flow problem. A scaling algorithm either solves a series of approximate versions of the original problem, or obtains a series of approximate solutions, with an increasing degree of accuracy. The cost scaling algorithm for the minimum cost network flow problem obtains a series of ϵ-optimal solutions with successively smaller values of ϵ. In each phase of the algorithm, the current ϵ-optimal solution is converted to an $\frac{\epsilon}{2}$-optimal solution. Each such phase is called an *ϵ-scaling* phase. When ϵ finally becomes less than $\frac{1}{n}$, the cost scaling algorithm terminates because the solution is optimal by Lemma 1 and Theorem 1.

Figure 1: The cost scaling algorithm

algorithm *cost_scaling*
begin
  set $\pi := 0$; $x := 0$; $\epsilon := C$;
  while $\epsilon \geq \frac{1}{n}$ **do**
    improve_approximation;
    set $\epsilon := \frac{\epsilon}{2}$;
  end while
end

We are now in a position to describe the cost scaling algorithm for the constrained maximum flow problem. The algorithm and its procedures are formally described in figures 1, 2 and 3. The cost scaling algorithm obtains a series of ϵ-optimal solutions with decreasing values of ϵ, starting with $\epsilon = C$. The algorithm then performs cost scaling phases by iteratively applying the *improve_approximation* procedure that transforms an ϵ-optimal solution into an $\frac{\epsilon}{2}$-optimal solution. At the end of every ϵ-scaling phase, the flow is an $\frac{\epsilon}{2}$-optimal solution to the constrained maximum flow problem, and at the end of the last ϵ-scaling phase, $\epsilon < \frac{1}{n}$. Thus, by Lemma 1, the algorithm terminates with an optimal solution to the constrained maximum flow problem.

At the beginning of an ϵ-scaling phase, the solution (x, π) is ϵ-optimal for the constrained maximum flow problem since the ϵ-optimality condition (Eq. 1) and the condition $cx = D$ are satisfied. The procedure *improve_approximation* converts this ϵ-optimal solution to an $\frac{\epsilon}{2}$-optimal solution (i) by converting the flow to an $\frac{\epsilon}{2}$-optimal pseudoflow, and then (ii) by converting the $\frac{\epsilon}{2}$-optimal pseudoflow to a flow and (iii) by restoring the condition $cx = D$ by sending flows through admissible paths from s to t. Thus, at the end of an ϵ-scaling phase, the pair (x, π) is an $\frac{\epsilon}{2}$-optimal solution to the constrained maximum flow problem.

The procedure *improve_approximation* first creates an $\frac{\epsilon}{2}$-optimal pseudoflow from an ϵ-optimal flow, but this may create imbalances at some nodes. The procedure then pushes flows from active nodes to inactive nodes to convert the pseudoflow into a flow. After each push, if $cx < D$, the procedure *find_admissible_path* identifies an admissible path and we push the

Figure 2: The procedure *improve_approximation* of the cost scaling algorithm

procedure *improve_approximation*;
begin
  for every arc $(i, j) \in A$ **do**
    if $c_{ij}^{\pi} > \frac{\epsilon}{2}$ **then**
      set $x_{ij} := 0$;
    else if $c_{ij}^{\pi} < -\frac{\epsilon}{2}$ **then**
      set $x_{ij} = u_{ij}$;
    end if
  end for
  while there is an active node in the network **do**
    pick an active node i;
    if there is an admissible arc (i, j) in $G(x)$ **then**
      push $\delta := \min\{e(i), r_{ij}\}$ units of flow from node i to node j;
    else
      set $\pi(i) := \pi(i) + \min_{j \in F(i)} c_{ij}^{\pi} + \frac{\epsilon}{2}$;
    end if
    while $cx < D$ **do**
      P := *find_admissible_path*;
      augment $\delta := \min\{\min_{(i,j)\in P} r_{ij}, (D - cx)/\sum_{(i,j)\in P} c_{ij}\}$ units of flow along P and (t, s);
    end while
  end while
end

required amount of flow from s to t to restore $cx = D$. If a push on an arc (i, j) is equal to its residual capacity r_{ij}, then we call it a *saturating* push; otherwise, it is called a *nonsaturating* push. When an active node contains no admissible arcs, we increase the node potential to create admissible arcs emanating from that node. Figure 2 formally describes the procedure *improve_approximation*, and Figure 3 describes the procedure *find_admissible_path*.

6 Analysis of the Algorithm

In this section, we will analyze the algorithm and prove that it terminates with an optimal solution to the constrained maximum flow problem and establish its worst case time complexity. The following lemma proves that the procedure *improve_approximation* starts with an ϵ-optimal solution and ends up with an $\frac{\epsilon}{2}$-optimal solution.

Lemma 2. *The procedure improve_approximation converts an ϵ-optimal solution to an $\frac{\epsilon}{2}$-optimal solution.*

Proof: At the beginning of the procedure, all arc flows are set to zero for arcs with reduced costs greater than $\frac{\epsilon}{2}$ and to their capacities for the ones with reduced costs less than $-\frac{\epsilon}{2}$.

Figure 3: The procedure *find_admissible_path* of the cost scaling algorithm

```
procedure find_admissible_path;
begin
  set P := ∅; i := s;
  while i ≠ t do
    j := First node in F(i) for which (i, j) is admissible;
    if (i, j) is admissible then
      {advance step}
      add (i, j) to P;
      set pred(j) := i; i := j;
    else
      {retreat & relabel step}
      set π(i) := π(i) + min_{j∈F(i)} c^π_ij + ε/2;
      if i ≠ s then
        remove arc (pred(i), i) from P;
        set i := pred(i);
      end if
    end if
  end while
  return P;
end
```

This makes all arcs that previously violate $\frac{\epsilon}{2}$-optimality satisfy it now. The arc flows with reduced costs that are within the interval $(-\frac{\epsilon}{2}, \frac{\epsilon}{2})$ already satisfy $\frac{\epsilon}{2}$-optimality, so they are not changed. The resulting pseudoflow is $\frac{\epsilon}{2}$-optimal. We will now show that *push*, *relabel* and *retreat & relabel* steps maintain $\frac{\epsilon}{2}$-optimality. When we push flow on an admissible arc (i, j), we may introduce the reverse arc (j, i) into $G(x)$. But since $-\frac{\epsilon}{2} \leq c^\pi_{ij} < 0$ (by the definition of admissibility), $c^\pi_{ji} = -c^\pi_{ij} > 0$, so the resulting pseudoflow will still satisfy $\frac{\epsilon}{2}$-optimality. If we increase the node potential $\pi(i)$ of a node i, reduced costs of all outgoing arcs will be lowered by an amount $\min_{j\in F(i)} c^\pi_{ij} + \frac{\epsilon}{2}$. Before the change, $c^\pi_{ij} \geq \min_{j\in F(i)} c^\pi_{ij}$ for all outgoing arcs (i, j), and after the change, $c^\pi_{ij} \geq -\frac{\epsilon}{2}$. Reduced costs of all incoming arcs will be increased by a positive amount, $\min_{j\in F(i)} c^\pi_{ij} + \frac{\epsilon}{2}$. Thus, $\frac{\epsilon}{2}$-optimality conditions are satisfied for all arcs after the node potential change. Push operations also convert the pseudoflow into a flow, and pushes from s to t restore $cx = D$; thus, the resulting flow is an $\frac{\epsilon}{2}$-optimal solution.□

We will now prove that the procedure *find_admissible_path* always terminates with an admissible path. The procedure *find_admissible_path* starts at node s and iteratively forms a path out of s, and it terminates when it reaches node t. Only if it cycles through a series of nodes, it will not terminate within a finite number of *advance* steps. But the admissible network stays acyclic as the following lemma establishes. Thus, the procedure always terminates in a finite number of steps.

Lemma 3. *The admissible network is acyclic throughout the cost scaling algorithm.*

Proof: This is true at the beginning of the algorithm as the initial reduced costs are all nonnegative and there are no admissible arcs in the residual network. We always push flow on an arc (i, j) when $c_{ij}^{\pi} < 0$, so if we add the reversal arc (j, i), $c_{ji}^{\pi} > 0$ and the reversal arc is not admissible. Thus, pushes do not add admissible arcs to the residual network and the admissible network stays acyclic. When we increase the node potential of node i, we create at least one admissible outgoing arc from i. However, since we increase $\pi(i)$ by $\min_{j \in F(i)} c_{ij}^{\pi} + \frac{\epsilon}{2}$, all incoming arcs of i become inadmissible. Thus, increasing the potential of a node cannot create a directed cycle passing through the node. Thus, neither pushes nor node potential increases can create directed cycles and the admissible network remains acyclic.□

We rely on the following lemma from Goldberg and Tarjan (1990) to establish a bound on the number of *relabel* and *retreat & relabel* steps throughout an execution of the procedure *improve_approximation*.

Lemma 4 (Goldberg and Tarjan (1990))**.** *Let x be a pseudoflow and x' be a flow. For any node v with $e(v) > 0$, there exists a node w with $e(w) < 0$ and a sequence of distinct nodes $v = v_0, v_1, \ldots, v_{l-1} = w$ such that $(v_i, v_{i+1}) \in G(x)$ and $(v_{i+1}, v_i) \in G(x')$ for $0 \leq i < l$.*

Proof: See Goldberg and Tarjan (1990), page 445.

Lemma 5. *During an execution of the procedure improve_approximation, the number of potential increases for any node is $O(n)$.*

Proof: Let x be the $\frac{\epsilon}{2}$-optimal pseudoflow, and x' be the ϵ-optimal flow at the end of the previous cost scaling phase, and π and π' be the corresponding node potentials. By Lemma 4, for every active node v, there exists a deficit node w and a directed path P from v to w in $G(x)$, where its reversal, P' is a directed path in $G(x')$. Then, arcs on P satisfy $\frac{\epsilon}{2}$-optimality and arcs on P' satisfy ϵ-optimality. Since a path can contain at most $n - 1$ arcs, it follows that:

$$\sum_{(i,j) \in P} c_{ij}^{\pi} \geq -(n-1)\frac{\epsilon}{2},$$

and

$$\sum_{(j,i) \in P'} c_{ji}^{\pi'} \geq -(n-1)\epsilon.$$

Substituting $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ and $c_{ji}^{\pi'} = -c_{ij} - \pi'(j) + \pi'(i)$ in the above expressions results in:

$$\sum_{(i,j) \in P} c_{ij} - \pi(v) + \pi(w) \geq -(n-1)\frac{\epsilon}{2}, \tag{6}$$

and

$$-\sum_{(i,j) \in P} c_{ij} - \pi'(w) + \pi'(v) \geq -(n-1)\epsilon. \tag{7}$$

Rearranging the terms in Eq. 7, we obtain:

$$(n-1)\epsilon - \pi'(w) + \pi'(v) \geq \sum_{(i,j) \in P} c_{ij}. \tag{8}$$

Substituting Eq. 8 in Eq. 6 results in:

$$(\pi(v) - \pi'(v)) \leq \frac{3}{2}(n-1)\epsilon + (\pi(w) - \pi'(w)). \tag{9}$$

Now observe that $\pi(w) = \pi'(w)$ throughout the algorithm, since we never increase node potentials for inactive nodes. Since the algorithm increases a node potential by at least $\frac{\epsilon}{2}$ units at a time, a node can be relabeled at most $3(n-1)$ times, or $O(n)$ times. □

Now, we are in a position to bound the number of saturating and nonsaturating pushes during an execution of the procedure *improve_approximation*. The following two lemmas establish those bounds.

Lemma 6. *The procedure improve_approximation performs O(nm) saturating pushes.*

Proof: Consider a saturating push of an arc (i, j). From the definition of admissibility, $c_{ij}^{\pi} < 0$ before the saturation. Before we can saturate this arc again, we must send back some flow on the reverse arc (j, i). In order to do that, (j, i) must be admissible, so $c_{ji}^{\pi} < 0$, or $c_{ij}^{\pi} > 0$. But this is only possible if we have relabeled node j. Then, in order to saturate (i, j) again, we must have $c_{ij}^{\pi} < 0$, which is only possible if we have relabeled node i. By Lemma 5, the number of relabelings for any node is $O(n)$. Thus, we can saturate any arc at most $O(n)$ times, which makes the total number of saturating pushes for the procedure *improve approximation* $O(nm)$. □

Lemma 7. *The procedure improve_approximation performs $O(n^2m)$ nonsaturating pushes.*

Proof: Let $f(i)$ be the number of nodes that are reachable from node i in the admissible network (including i). Let Φ be $\sum_{i:\ active} f(i)$. Throughout the procedure, there can be at most $(n-1)$ active nodes in the network since there must be at least one deficit node. At any time, $f(i) \leq n$ for each active node, which bounds Φ by $O(n^2)$. A saturating push on an arc (i, j) might make node j active, which would increase Φ by $f(j) \leq n$. Lemma 6 implies that the total increase in Φ due to saturating pushes is $O(n^2m)$. A relabeling operation on node i may create new admissible arcs, which may increase $f(i)$ by at most n units. Since relabeling of node i makes all incoming arcs of i inadmissible, $f(k)$ will not change for any other node k. Lemma 5 implies that the total increase in Φ due to relabeling operations is $O(n^3)$. Now, consider the effect of nonsaturating pushes on Φ. After a nonsaturating push on arc (i, j), node i becomes inactive and node j might become active. Thus, Φ may increase by at most $f(j) - f(i)$ units after a nonsaturating push. But $f(i) \geq f(j) + 1$ since every node that is reachable from j is also reachable from i and by Lemma 3, node j is not reachable from node i. Thus, Φ decreases by at least one unit after a nonsaturating push. Thus, the number of nonsaturating pushes throughout the procedure *improve_approximation* is bounded by: $O(n^2) + O(n^2m) + O(n^3) = O(n^2m)$. □

Lemma 8. *The procedure improve_approximation runs in $O(n^2m)$time.*

Proof: By Lemma 6, the number of saturating pushes is $O(nm)$, and by Lemma 7, the number of nonsaturating pushes is $O(n^2m)$. A push from s to t consists of saturating and nonsaturating pushes, thus the total time for those is also bounded by $O(n^2m + nm)$. An execution of the procedure *find_admissible_path* consists of *advance* and *retreat & relabel* steps. By Lemma 5, the number of *retreat & relabel* steps is $O(n)$. The *advance* steps can be classified into two types: the ones that are later cancelled, and the ones that are not cancelled. The first type are the *retreat* part of the *retreat & relabel* steps, thus they are $O(n)$. The second type are also $O(n)$ because the length of a path cannot be more than $n-1$. Finally, the total time to find an admissible arc for all nodes in the network before one relabeling is $O(m)$, as one only has to go through every outgoing arc once before a relabeling. Since each node is relabeled at most $O(n)$ times, the total time for finding admissible arcs is $O(nm)$. Thus, the overall complexity of the procedure *improve_approximation* is: $O(n^2m + nm + n^2m + nm + n + n + nm) = O(n^2m)$. □

We are now in a position to determine the overall complexity of the cost scaling algorithm for the constrained maximum flow problem. The algorithm starts with an ϵ value of C, and it halves its value until $\epsilon < \frac{1}{n}$. Therefore, the algorithm performs $1 + logC - log(\frac{1}{n}) = 1 + log(nC) = O(log(nC))$ cost scaling phases. We have thus established the following theorem:

Theorem 2. *The cost scaling algorithm for the constrained maximum flow problem runs in* $O(n^2mlog(nC))$ *time.*

7 Conclusion

In this research we propose a polynomial combinatorial algorithm for the constrained maximum flow problem that runs in $O(n^2m\log(nC))$ time. The constrained maximum flow problem is important to study as it has many applications in many areas of business and engineering. This worst case time bound is competitive against the existing polynomial algorithms in the literature: the capacity scaling and the double scaling algorithms. Future research will focus on the empirical performance of the new algorithm compared to the existing ones.

References

R.K. Ahuja and J.B. Orlin. A capacity scaling algorithm for the constrained maximum flow problem. *Networks*, 25:89–98, 1995.

R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms and applications*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

D.P. Bertsekas. A distributed algorithm for the assignment problem. Working Paper, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1979.

C. Çalışkan. A double scaling algorithm for the constrained maximum flow problem. *Computers and Operations Research*, 35(4):1138–1150, 2008.

A. V. Goldberg and R. E. Tarjan. Solving mimimum cost flow problem by successive approximation. In *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pages 7–18, 1987. Full paper in *Mathematics of Operations Research* 15(1990), 430-466.

A. V. Goldberg and R. E. Tarjan. Finding mimimum cost flow circulations by successive approximation. *Mathematics of Operations Research*, 15:430–466, 1990.

H. Röck. Scaling Techniques for Minimal Cost Network Flows, pages 181–191. Discrete Structures and Algorithms. Carl Hanser, Munich, 1980.

É Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5: 247–155, 1985.