A COMPUTATIONAL STUDY OF THE CONSTRAINED MAXIMUM FLOW PROBLEM

Cenk Çalışkan, Woodbury School of Business, Utah Valley University, 800 W. University Pkwy, Orem, UT 84058, (801) 863-6487, cenk.caliskan@uvu.edu

ABSTRACT

The constrained maximum flow problem is to send the maximum flow from a source node to a sink node in a directed capacitated network for which the total cost of the flow is constrained by a budget limit. It has many applications in telecommunications, logistics and computer networks. We present a new polynomial cost scaling algorithm and compare its computational time against the existing polynomial combinatorial algorithms. We show that the proposed algorithm is on average 25 times faster than the double scaling algorithm, and 32 times faster than the capacity scaling algorithm.

INTRODUCTION

The constrained maximum flow problem is a variant of the maximum flow problem. Many algorithms were proposed to solve the maximum flow problem, but very few articles appeared in the literature for its variants. Ahuja and Orlin [2] 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. Çalışkan [4] 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 tree*s due to Goldberg and Tarjan [5]. In this research, we present a cost scaling algorithm that runs in $O(n^2 m\log(nC))$ time and provide the results of computational tests that show that the new algorithm significantly outperforms the existing polynomial algorithms.

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 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. Without loss of generality, 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, as transformations in Ahuja and Orlin [2] could be applied to satisfy them.

The problem then can be formulated as follows:

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

s.t.

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

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

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

Theorem 1 (Ahuja and Orlin [2]) *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 [2], page 91.

By Theorem 1, we can modify any algorithm for the minimum cost network flow problem to solve the constrained maximum flow problem. Our cost scaling algorithm for the constrained maximum flow problem is based on this idea.

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 call $F(i,G) = \{j \in N \mid (i,j) \in G\}$ the *forward star* of i in G. Similarly, for each node $i \in N$, we call $B(i,G) = \{j \in N \mid (j,i) \in G\}$ the *backward star* of i in G. The index of the node preceding a given node i in a path is denoted by $pred(i)$.

A *flow* is a function $x : A \to R^+ \cup \{0\}$ that satisfies Eqs. 2 and 4. The value of x_{ts} is called the value of flow x. If a flow x also satisfies Eq. 3, it is called a *feasible flow*. A *pseudoflow* is a function $x : A \to R^+ \cup \{0\}$ that satisfies only Eq. 4. For a pseudoflow x, we define the imbalance of node $i \in N$ as follows: $e(i) = \sum_{(j,i)\in A} x_{ji} - \sum_{(i,j)\in A} x_{ij}$.

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 [3], and independently, Tardos [6].

Definition 1 (Epsilon Optimality) *A flow x or a pseudoflow x is called ε-optimal for some ε and π, if $c_{ij}^{\pi} \geq -\varepsilon$ 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 $\varepsilon = 0$. When the arc costs are integers, any ε-optimal flow is optimal for $\varepsilon < 1/n$, as the following lemma establishes:

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

Proof: See Ahuja et al. [2], page 363.

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 $-\varepsilon/2 \leq c_{ij}^{\pi} < 0$, 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.

THE COST SCALING ALGORITHM

The algorithm and its procedures are formally described in Figures 1, 2 and 3. In another paper, we show that the algorithm runs in $O(n^2 m \log(nC))$ time. The cost scaling algorithm obtains a series of ε-optimal solutions with decreasing values of ε, starting with $\varepsilon = C$. The algorithm then performs cost scaling phases by iteratively applying the *improve_approximation* procedure that transforms an ε-optimal solution into an $\varepsilon/2$-optimal solution. At the end of every ε-scaling phase, the flow is an $\varepsilon/2$-optimal solution to the constrained maximum flow problem, and at the end of the last ε-scaling phase, $\varepsilon < 1/n$. Thus, by Lemma 1, the algorithm terminates with an optimal solution to the constrained maximum flow problem.

```
algorithm cost_scaling
begin
    set π := 0; x := 0; ε := C;
    while ε ≥ 1/n do
        improve_approximation;
        set ε := ε/2;
    end while
end
```

Figure 1: The cost scaling algorithm

At the beginning of an ε-scaling phase, the solution (x,π) is ε-optimal for the constrained maximum flow problem by Definition 1 and Theorem 1. Procedure *improve_approximation* converts this ε-optimal solution to an $\varepsilon/2$-optimal solution (i) by converting the flow to an $\varepsilon/2$-optimal pseudoflow, and then (ii) by converting the $\varepsilon/2$-optimal pseudoflow to a flow and (iii) by restoring the condition $cx = D$ via flow augmentations on admissible paths from s to t. Thus, at the end of an ε-scaling phase, (x,π) is an $\varepsilon/2$-optimal solution to the constrained maximum flow problem.

procedure *improve_approximation*;
begin
  for every arc $(i,j) \in A$ **do**
    if $c_{ij}^{\pi} > \varepsilon/2$ **then**
      set $x_{ij} := 0$;
    else if $c_{ij}^{\pi} < -\varepsilon/2$ **then**
      set $x_{ij} = u_{ij}$;
    end if
  end for
  while there is an active node i in the network **do**
    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 $\pi(i) := \pi(i) + \min_{j \in F(i)} c_{ij}^{\pi} + \varepsilon/2$;
    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

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

The procedure *improve_approximation* first creates an $\varepsilon/2$-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 required amount of flow from s to t to restore $cx = D$. 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*.

COMPUTATIONAL RESULTS

In order to test the empirical performance of the cost scaling algorithm, we generated problem instances using a random network generator similar to the one that was described in Çalışkan [4] and compared the cost scaling algorithm to the capacity scaling and the double scaling algorithms. All algorithms were

```
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
         add (i, j) to P;
         set pred(j) := i; i := j;
      else
         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
```

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

coded in the same programming style, using the same network representation and data structures, so that there was no performance variation due to differences in implementation. We coded all algorithms in C++ and compiled with Microsoft Visual C++ 7.1, using the optimization option /O2. We conducted the runs on a computer with 2.0 GHz Intel Core 2 Duo processor and 2.0 GB of memory. We generated networks that have up to 16384 nodes and 524288 arcs in our experiments. The arc capacities were uniformly distributed in $[1,10^4]$ and the arc costs were uniformly distributed in $[1,10^2]$. We generated 10 random instances for each combination of the parameters. Table 1 shows the average CPU times.

The results in Table 1 show that the cost scaling algorithm is computationally superior. In the experiments, the cost scaling algorithm was up to 56 times faster than the double scaling algorithm with an average of 25 times faster; and up to 173 times faster than the capacity scaling algorithm with an average of 32 times faster. Furthermore, it was significantly faster than both algorithms for every instance of the test problems, with a minimum of 19 times faster than the double scaling algorithm, and a minimum of 7 times faster than the capacity scaling algorithm. Thus, the cost scaling algorithm is empirically the fastest combinatorial polynomial algorithm for the constrained maximum flow problem.

CONCLUSION

In this research we propose a polynomial combinatorial algorithm for the constrained maximum flow problem that runs in $O(n^2 m \log(nC))$ time. The constrained maximum flow problem is important to study as it has many applications and it is related to some important classical combinatorial optimization problems. Our computational tests show that the proposed algorithm is significantly faster than the existing combinatorial polynomial algorithms for the problem: on average, 25 times faster than the double scaling algorithm, and 32 times faster than the capacity scaling algorithm.

Network Size		CPU times (sec.)			CPU Time Ratios	
		Cost	Double	Capacity		
n	m	Scaling (i)	Scaling (ii)	Scaling (iii)	(ii/i)	(iii/i)
256	2048	0.012	0.470	0.140	38	11
512	4096	0.060	1.495	0.495	25	8
1024	8192	0.188	4.258	2.466	23	13
2048	16384	0.773	14.544	10.857	19	14
4096	32768	2.895	67.509	93.971	23	32
8192	65536	13.111	297.225	611.961	23	47
16384	131072	155.013	†	†	-	-
256	4096	0.017	0.941	0.208	56	12
512	8192	0.078	2.206	0.569	28	7
1024	16384	0.272	7.487	7.607	27	28
2048	32768	1.110	26.189	37.608	24	34
4096	65536	4.352	104.308	294.300	24	68
8192	131072	19.102	490.647	1341.136	26	70
16384	262144	242.528	†	†	-	-
256	8192	0.041	1.923	0.413	47	10
512	16384	0.122	4.603	1.639	38	13
1024	32768	0.480	14.395	13.725	30	29
2048	65536	1.687	49.539	189.025	29	112
4096	131072	6.298	167.725	1088.305	27	173
8192	262144	26.549	670.588	†	25	-
16384	524288	325.738	†	†	-	-

† The program was stopped after 1400 seconds elapsed

Table 1: CPU times of the cost scaling, double scaling and capacity scaling algorithms

REFERENCES

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

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

[3] Bertsekas, D.P. A Distributed Algorithm for the Assignment Problem. Working Paper, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1979.

[4] Çalışkan, C. A Double Scaling Algorithm for the Constrained Maximum Flow Problem. *Computers and Operations Research*, 2008, 35(4), 1138-1150.

[5] Goldberg, A. V. and Tarjan, R. E. Finding Mimimum Cost Flow Circulations by Successive Approximation. *Mathematics of Operations Research*, 1990, 15, 430-466.

[6] Tardos, É. A Strongly Polynomial Minimum Cost Circulation Algorithm. *Combinatorica*, 1985, 5, 247-155.