

A Capacity Scaling Algorithm for
The Constrained Maximum Flow Problem

Ravindra K. Ahuja
I. I. T. Kanpur
and
James B. Orlin, MIT

WP# 3587-93-MSA July 1993

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE JUL 1993		2. REPORT TYPE		3. DATES COVERED 00-07-1993 to 00-07-1993	
4. TITLE AND SUBTITLE A Capacity Scaling Algorithm for the Constrained Maximum Flow Problem				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology,Laboratory for Information and Decision Systems,77 Massachusetts Avenue,Cambridge,MA,02139				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 22	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

A CAPACITY SCALING ALGORITHM
FOR
THE CONSTRAINED MAXIMUM FLOW PROBLEM

Ravindra K. Ahuja
Department of Industrial and Management Engineering
Indian Institute of Technology
Kanpur - 208016, INDIA

James B. Orlin
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

A CAPACITY SCALING ALGORITHM FOR THE CONSTRAINED MAXIMUM FLOW PROBLEM

Ravindra K. Ahuja and James B. Orlin

ABSTRACT

The constrained maximum flow problem is to send the maximum possible flow from a source node s to a sink node t in a directed network subject to a budget constraint that the cost of flow is no more than D . In this paper, we consider two versions of this problem: (i) when the cost of flow on each arc is a linear function of the amount of flow; and (ii) when the cost of flow is a convex function of the amount of flow. We suggest capacity scaling algorithms that solve both versions of the constrained maximum flow problem in $O((m \log M) S(n, m))$ time, where n is the number of nodes in the network, m is the number of arcs, M is an upper bound on the largest element in the data, and $S(n, m)$ is the time required to solve a shortest path problem with nonnegative arc lengths. Our algorithms are modifications of the capacity scaling algorithms for the minimum cost flow and convex cost flow problems, and illustrate the power of capacity scaling algorithms to solve variants of the minimum cost flow problem in polynomial time.

1. INTRODUCTION

Let $G = (N, A)$ be a directed network consisting of a set N of nodes and a set A of arcs. In this network, each arc (i, j) has an associated cost c_{ij} and a capacity u_{ij} . The linear cost constrained maximum flow problem is to send the maximum possible flow from a source node s to a sink node t subject to the cost of flow being less than or equal to a budget D . This problem can be formulated as the following linear program:

$$\text{Maximize } v \quad (1a)$$

subject to

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = \begin{cases} v & \text{for } i = s \\ 0 & \text{for all } i \in N - \{s \text{ and } t\} \\ -v & \text{for } i = t \end{cases} \quad (1b)$$

$$0 \leq x_{ij} \leq u_{ij} \text{ for } (i, j) \in A, \quad (1c)$$

$$\sum_{(i,j) \in A} c_{ij} x_{ij} \leq D. \quad (1d)$$

Similarly, we can define the convex cost constrained maximum flow problem. In this problem, the cost of flow on each arc $(i, j) \in A$ is $C_{ij}(x_{ij})$, which is a convex function of the flow x_{ij} . The formulation of this problem is the same as that of the linear cost constrained maximum flow problem given in (1) except that (1d) is replaced by the following constraint:

$$\sum_{(i,j) \in A} C_{ij}(x_{ij}) \leq D. \quad (1e)$$

We shall now focus on the linear cost constrained maximum flow problem and postpone the discussion on the convex cost constrained maximum flow problem until Section 5.

Let $n = |N|$ denote the number of nodes in the network, $m = |A|$ denote the number of arcs, C denotes the largest arc cost, and U denote the largest of the finite arc capacities in the network. We consider the linear cost constrained maximum flow problem subject to the following three assumptions:

Assumption 1 (Integrality assumption). All arc capacities and arc costs are integers.

Assumption 2 (Connectivity assumption). The network is strongly connected; i.e., there is a directed path of sufficiently large capacity between every pair of nodes.

Assumption 3 (Nonnegative cost assumption). *Each arc cost is a nonnegative integer, and every directed path from the source node s to the sink node t has a cost greater than zero.*

We point out that there is some loss of generality in the first assumption and our proposed algorithm does really require that arc capacities are integral. Notice, however, that rational arc capacities can be made integral by multiplying them by a suitably large number. We also point out that we can satisfy the connectivity assumption by adding the arcs (s, i) and (i, s) for each node $i \in N - \{s\}$ of infinite capacity with cost $D + 1$; since these arcs have a high cost, no feasible solution can have a positive flow on these arcs. In Section 4, we explain how to transform a problem with negative costs into an equivalent problem satisfying Assumption 3. It follows from the integrality and nonnegative cost assumptions that D is an upper bound on the maximum amount of flow that can be sent from node s to node t .

Fulkerson [1959] describes an interesting application of the constrained maximum flow problem arising in the capacity expansion of a network. A network is used to send flow from node s to node t , and the arc capacities are insufficient for meeting anticipated future demands. Suppose that we can purchase additional capacities of some arcs at a cost of c_{ij} per unit increase in the capacity of arc (i, j) . Suppose further that we have an available budget of D units for purchasing additional capacities. We want to purchase additional capacities so as to keep the cost of expansion within the budget and such that the maximum flow from node s to node t is as large as possible. It is easy to observe that this problem is an instance of the convex cost constrained maximum flow problem where the cost of flow on arc (i, j) remains zero as long as $x_{ij} \leq u_{ij}$ but then increases at the rate of c_{ij} per unit additional flow on the arc.

The linear cost constrained maximum flow problem is very closely related to the minimum cost flow problem. We shall show that the well-known successive shortest path algorithm can be used to solve the constrained maximum flow problem in pseudo-polynomial time. One can also develop a binary search algorithm that solves the constrained maximum flow problem within $O(\log M)$ applications of any minimum cost flow algorithm. This approach yields a polynomial-time algorithm for the constrained maximum flow problem if we use a polynomial-time minimum cost flow algorithm as a subroutine. Currently, the best available time bound to solve the minimum cost flow problem is $O(\min\{nm \log(n^2/m) \log(nC), nm(\log \log U) \log(nC), m \log n (m + n \log n)\})$, and the three time bounds in this expression are, respectively, due to Goldberg and Tarjan [1987], Ahuja et al. [1992], and Orlin [1988].

Since most of the available polynomial-time algorithms for the minimum cost flow problem are scaling based algorithms, a natural question arises whether we can modify any scaling algorithm for the minimum cost flow problem so that it solves the constrained maximum flow problem in the same time. In this paper, we answer this question in the affirmative for capacity scaling algorithms and modify Edmonds and Karp's [1972] capacity scaling algorithm

for the minimum cost flow problem so that it solves the constrained maximum flow problem in the same time (see also Ahuja, Magnanti and Orlin [1993]). This approach yields an $O(m \log M S(n, m))$ time algorithm to solve the constrained maximum flow problem, where $S(n, m)$ is the time needed to solve a shortest path problem with nonnegative arc lengths. Currently, $S(n, m) = \min\{m + n \log n, m + n (\log C)^{1/2}, m \log \log C\}$, where C is the largest arc cost encountered in the shortest path problem; the three time bounds in this expression are due to Fredman and Tarjan [1984], Ahuja et al. [1990], and Johnson [1982]. For some classes of minimum cost flow problems, this approach provides the fastest available algorithm to solve the constrained maximum flow problem. We also generalize this algorithm to solve the convex cost constrained maximum flow problem. This generalized algorithm obtains an integer optimal solution of the convex cost constrained maximum flow problem in the same time as for the linear cost case.

2. RELATIONSHIP TO THE MINIMUM COST FLOW PROBLEM

The constrained maximum flow problem is closely related to the minimum cost flow problem and understanding this relationship is essential for its algorithmic development. In this section, we study the relationship between these two problems.

One version of the minimum cost flow problem is to determine the least cost shipment of v units of a commodity from the source node s to the sink node t . This minimum cost flow problem can be formulated as the following linear programming problem:

$$\text{Minimize} \quad \sum_{(i,j) \in A} c_{ij} x_{ij}, \quad (2a)$$

subject to

$$\sum_{\{j:(i,j) \in A\}} x_{ij} - \sum_{\{j:(j,i) \in A\}} x_{ji} = \begin{cases} v & \text{for } i = s \\ 0 & \text{for all } i \in N - \{s \text{ and } t\} \\ -v & \text{for } i = t \end{cases}, \quad (2b)$$

$$0 \leq x_{ij} \leq u_{ij} \text{ for all } (i, j) \in A. \quad (2c)$$

Our algorithms rely on the concept of residual networks. The residual network $G(x)$ corresponding to a flow x (for the the minimum cost flow problem as well as for the constrained maximum flow problem) is defined as follows: We replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i) . The arc (i, j) has cost c_{ij} and *residual capacity* $r_{ij} = u_{ij} - x_{ij}$, and the arc (j, i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = x_{ij}$. The residual network consists *only* of arcs with positive residual capacity.

In this formulation of the minimum cost flow problem, we associate a *dual variable* $\pi(i)$ with the mass balance constraint (2b) of node i ; we refer to $\pi(i)$ as the *potential* of node i . With respect to a set of node potentials π , we define the *reduced cost* c_{ij}^π of an arc (i, j) as $c_{ij}^\pi = c_{ij} - \pi(i)$

+ $\pi(j)$. In our subsequent discussion, we shall make use of the following well-known optimality conditions for the minimum cost flow problem (see, for example, Ahuja, Magnanti and Orlin [1993]).

Optimality Conditions. *A feasible solution x^* of the minimum cost flow problem is an optimal solution if and only if there exists a set of node potentials π satisfying the following optimality conditions:*

$$c_{ij}^{\pi} \geq 0 \text{ for every arc } (i, j) \text{ in } G(x^*). \quad (3)$$

We now prove a result that establishes a close relationship between the minimum cost flow problem and the constrained maximum flow problem. We use this relationship to prove the correctness of our proposed algorithms for the latter problem.

Theorem 1. *Let x^* be an optimal solution of the minimum cost flow problem when the supply of the source node is constrained to be equal to v^* . Then x^* is also an optimal solution of the constrained maximum flow problem with solution value v^* if $D = cx^*$.*

Proof. We prove this result by contradiction. Suppose that x^* is not an optimal solution of the constrained maximum flow problem with $D = cx^*$. Instead, x^1 is an optimal solution with $v^1 > v^*$ and $cx^1 \leq D$. According to the flow decomposition theory (see, for example, Ahuja, Magnanti and Orlin [1993]), the flow x^1 can be decomposed into flows along directed paths from node s to node t and flows along cycles. From Assumption 3, each directed path from node s to node t has a positive length. Therefore, starting with the flow x^1 , we can reduce the flow along some directed paths from node s to node t and obtain a flow x^2 of lesser cost whose flow value equals $v^2 = v^*$. This implies that x^2 has a flow value v^* and its cost is $cx^2 < cx^1 \leq D = cx^*$. This contradicts our assumption that x^* is an optimal solution of the minimum cost flow problem with flow value v^* . ♦

This theorem has several implications. It shows a connection between the minimum cost flow problem and the constrained maximum flow problem and allows us to solve the latter problem by an algorithm for the former problem. Suppose that we have an algorithm for the minimum cost flow problem that parametrically increases flow from node s to node t , i.e., we find the minimum cost flow when the supply v^* at the source node is parametrically increased from 0. If we apply this algorithm and terminate its execution when either the maximum possible flow has been obtained from node s to node t or the flow cost exactly equals D , then the resulting solution is an optimal solution of the constrained maximum flow problem.

Another implication of Theorem 1 is that one can readily solve the constrained maximum flow problem when the budget constraint is redundant. The budget constraint is said to be *redundant* if and only if there is a maximum flow from node s to node t that satisfies the

constraint (1d). We can determine the redundancy of the budget constraint using the following method. We first solve a maximum flow problem to determine the maximum possible flow, say v^0 , that can be sent from node s to node t . We then solve a minimum cost flow problem with $v = v^0$. If the cost of the optimal solution is less than or equal to D , then the budget constraint is redundant, and non-redundant otherwise. For the simplicity of exposition, we shall henceforth assume that the budget constraint is always nonredundant, i.e., is a *binding constraint*.

Assumption 4 (Tight budget assumption). *In the constrained maximum flow problem (1), every optimal solution x^* satisfies $cx^* = D$.*

Theorem 1 also implies that we can solve the constrained maximum flow problem using binary search and using any minimum cost flow algorithm as a subroutine. In the minimum cost flow problem, we can perform a binary search on the integer flow values v and determine the minimum cost flow solutions x^1 and x^2 whose integer flow values satisfy v^1 and v^2 satisfying (i) $v^2 = v^1 + 1$, and (ii) $cx^1 \leq D \leq cx^2$. Then the optimal flow value v^* satisfies $v^1 \leq v^* \leq v^2$. We determine v^* and its associated arc flow x^* in the following manner. We determine shortest path distances $d(\cdot)$ from node s to all other nodes in the residual network $G(x^1)$ and augment $(D - cx^1)/d(t)$ units of flow from node s to node t along the shortest path. The resulting flow is an optimal flow of the constrained maximum flow problem. (For justification, see Lemma 1.) This approach requires solving $O(\log D)$ minimum cost flow problems, because the flow values v tested by the binary search technique lie in the range $[0, D]$. When we allow the constrained maximum flow problem to have negative arc lengths and perform the transformation described in Section 4, then the binary search technique would require solving $O(\log M)$ minimum cost flow problems, where $M = \max\{n, m, C, U, D\}$. The running time of this approach is $O((\log M)(\min\{nm \log(n^2/m) \log(nC), nm(\log \log U) \log(nC), m \log n (m + n \log n)\}))$.

We point out that the optimal solution of the constrained maximum flow problem may not be integer. However, if one adds the additional constraint that the flows be integral, then an optimal solution of the modified problem is easy to obtain. Suppose that x^* is a (real) optimal solution of the constrained maximum flow problem with flow value v^* . It can be easily verified that the solution x' obtained by solving a minimum cost flow problem (2) with flow value $v' = \lfloor v^* \rfloor$ is an optimal solution of the constrained maximum flow problem, requiring flows to be integral. We refer to x' as an optimal integer flow of the constrained maximum flow problem.

3. SUCCESSIVE SHORTEST PATH ALGORITHM

The successive shortest path algorithm is a well-known algorithm to solve the minimum cost flow problem, and by modifying it slightly we can solve the constrained maximum flow problem. This (modified) successive shortest path algorithm forms the basis of

the scaling algorithms for linear and convex cost constrained maximum flow problems. In this section, we present a brief description of the successive shortest path algorithm and its proof of correctness. For a more detailed description of this algorithm, we refer the reader to Ahuja, Magnanti and Orlin [1993].

In order to describe this algorithm as well as several later developments, we first introduce the concept of pseudoflows. A pseudoflow is a function x defined on arcs that satisfies only the capacity and nonnegativity constraints; it need not satisfy the mass balance constraints. For any pseudoflow x , we define the *imbalance* of a node i as

$$e(i) = b(i) + \sum_{\{j:(j,i) \in A\}} x_{ji} - \sum_{\{j:(i,j) \in A\}} x_{ij} \text{ for all } i \in N. \quad (4)$$

If $e(i) > 0$ for some node i , then we refer to $e(i)$ as the *excess* of node i ; if $e(i) < 0$, then we refer to $-e(i)$ as the node's *deficit*. We refer to a node i with $e(i) = 0$ as *balanced*. The residual network for a pseudoflow is defined in the same way that we define the residual network for a flow.

The successive shortest path algorithm for the constrained maximum flow problem maintains a special type of pseudoflow, for which imbalances at all nodes, except the source and sink nodes, are zero. Further, this pseudoflow satisfies the optimality conditions (3). At each step, the algorithm identifies a shortest path in the residual network from node s to node t and augments the maximum possible flow along the path. The algorithm also keeps track of the cost of flow and terminates when the cost of flow equals D . Figure 1 gives an algorithmic description of the successive shortest path algorithm.

```

algorithm successive shortest path;
begin
   $x := 0$ ;  $\pi := 0$ ;  $v := 0$ ;
  determine shortest path distances  $d(\cdot)$  from node  $s$  using  $c^\pi$  as arc lengths
    and update  $\pi := \pi - d$ ;
  let  $P$  be the shortest path from node  $s$  to node  $t$ ;
  while  $(D - cx) > 0$  do
    begin
      compute  $\delta := \min[\min\{r_{ij} : (i, j) \in P\}, (D - cx)/(-\pi(t))]$ ;
      augment  $\delta$  units of flow along the path  $P$  in  $G(x)$  from node  $s$  to node  $t$ ;
      update  $x$ ,  $cx$ ,  $v$ , and  $G(x)$ ;
      determine shortest path distances  $d(\cdot)$  from node  $s$  using
         $c^\pi$  as arc lengths and update  $\pi := \pi - d$ ;
      let  $P$  be the shortest path from node  $s$  to node  $t$ ;
    end;
  end;

```

Figure 1. The successive shortest path algorithm for the constrained maximum flow problem.

Since the solution satisfies the optimality conditions $c_{ij}^{\pi} \geq 0$ for every arc (i, j) in the residual network, we can use any shortest path algorithm for the nonnegative arc lengths to obtain the shortest path distances $d(\cdot)$. The algorithm then updates the node potentials as $\pi' = \pi - d$. Since the vector d represents the shortest path distances with c_{ij}^{π} as the arc lengths in the residual network $G(x)$, it satisfies the shortest path optimality conditions, i.e., $d(j) \leq d(i) + c_{ij}^{\pi}$ for all (i, j) in $G(x)$ (see, e.g., Ahuja, Magnanti and Orlin [1993]). Substituting $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ in the preceding inequality, we obtain $d(j) \leq d(i) + c_{ij} - \pi(i) + \pi(j)$. Alternatively, $c_{ij} - (\pi(i) - d(i)) + (\pi(j) - d(j)) \geq 0$, or $c_{ij}^{\pi'} \geq 0$. This establishes that the pseudoflow x satisfies the optimality conditions with respect to the node potentials $\pi' = \pi - d$, and in the next iteration the reduced arc costs are again nonnegative. Now consider a shortest path P from node s to node t . For each arc (i, j) in this path, $d(j) = d(i) + c_{ij}^{\pi}$. Substituting $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$ in this equation, we obtain $c_{ij}^{\pi'} = 0$. In other words, every arc in the path P has a zero reduced cost. Augmenting flow on any such arc might add its reversal (j, i) to the residual network. But since $c_{ij}^{\pi'} = 0$ for each arc $(i, j) \in P$, $c_{ji}^{\pi'} = 0$, and the arc (j, i) also satisfies the reduced cost optimality condition (3). Hence the following result:

Lemma 1. *Suppose that a pseudoflow (or a flow) x satisfies the optimality conditions with respect to the potentials π and the vector d denote the shortest path distances from node s (or node k) to all other nodes with respect to the arc lengths c_{ij}^{π} . Then the following properties hold:*

- (a) *The pseudoflow x satisfies the reduced cost optimality conditions with respect to the potentials $\pi' = \pi - d$.*
- (b) *If we obtain x' from x by sending flow along a shortest path from node s (or node k) to node t (or node l), then x' satisfies the optimality conditions with respect to the potentials π' . ♦*

We also point out that $(-\pi'(t))$ is the cost of sending one unit of flow from node s to node t along the shortest path P . This can be easily observed by using (i) $\pi'(s) = 0$; and (ii) $c_{ij}^{\pi'} = c_{ij} - \pi'(i) + \pi'(j) = 0$ for every arc $(i, j) \in P$. Also notice that in the algorithm the flow is integral in all iterations except the last iteration. In the last iteration, however, if we augment $\lfloor \delta \rfloor$ units of flow instead of δ units, then we get an optimal integer flow. The preceding lemma establishes that the successive shortest path algorithm always maintains a solution that satisfies the minimum cost flow optimality conditions. At termination, this solution x satisfies $cx = D$. It follows from Theorem 1 that the solution x is an optimal solution of the constrained maximum flow problem.

The successive shortest path algorithm is quite a straightforward algorithm to solve the constrained maximum flow problem; however, an important theoretical limitation is that

it doesn't run in polynomial time. In the next section, we develop a capacity scaling version of this algorithm which makes it a polynomial-time algorithm.

4. CAPACITY SCALING ALGORITHM

In this section, we present a capacity scaling algorithm to solve the constrained maximum flow problem. This algorithm is a scaling version of the successive shortest path algorithm described in Section 3 and borrows ideas from the variants of the capacity scaling algorithm for the minimum cost flow problem described in Orlin [1988], and Ahuja, Magnanti and Orlin [1993].

A scaling algorithm typically solves a series of approximate versions of the original problem and the degree of approximation gradually improves. A capacity scaling algorithm approximates arc capacities to varying degrees of accuracy in stages, called *scaling phases*. Each scaling phase has an associated value Δ of a parameter, which is a suitable power of 2, and we refer to a specific scaling phase as the Δ -scaling phase. In the Δ -scaling phase, we denote arc capacities by $u_{ij}(\Delta)$ and define them as per the following formula:

$$u_{ij}(\Delta) = \left\lfloor \frac{u_{ij}}{\Delta} \right\rfloor \Delta \quad \text{for every arc } (i, j) \in A. \quad (5)$$

In other words, we define $u_{ij}(\Delta)$ as the greatest multiple of Δ less than or equal to u_{ij} . For example, if $u_{ij} = 13$, then $u_{ij}(16) = 0$; $u_{ij}(8) = 8$; $u_{ij}(4) = 12$; $u_{ij}(2) = 12$; and $u_{ij}(1) = 13 = u_{ij}$. The following property is immediate from this definition.

Lemma 2. For every arc $(i, j) \in A$, if $2^k > u_{ij}$, then $0 = u_{ij}(2^k) \leq u_{ij}(2^{k-1}) \leq \dots \leq u_{ij}(2) \leq u_{ij}(1) = u_{ij}$. ♦

We refer to the constrained maximum flow problem with $u_{ij}(\Delta)$ as arc capacities as the Δ -scaled problem. When defining the residual network, if we replace the arc capacities u_{ij} by $u_{ij}(\Delta)$, then the resulting residual network is called the Δ -residual network. We denote the Δ -residual network by $G(x, \Delta)$. Notice that in the Δ -residual network, each residual capacity is a multiple of Δ .

We are now in a position to describe the capacity scaling algorithm. The capacity scaling algorithm solves a sequence of Δ -scaled problems with decreasing values of Δ . But instead of solving each such problem exactly, it solves it approximately and obtains a Δ -optimal solution. We refer to a solution of the Δ -scaled problem as a Δ -optimal solution if (i) it satisfies the optimality conditions (3); (ii) all arc flows are integral multiples of Δ ; and (iii) sending Δ additional units from node s to node t along the shortest path in $G(x, \Delta)$ violates the budget constraint. In other words, a Δ -optimal solution solves the Δ -scaled problem subject to the additional constraint that all arc flows are multiples of Δ . Therefore, a 1-optimal solution is an optimal integral flow for the constrained maximum flow problem. Let x^1 be an 1-optimal

solution of the constrained maximum flow problem. We can convert this 1-optimal solution into a real-valued optimal solution by determining the shortest path distances $d(\cdot)$ from node s to all other nodes in $G(x^1)$ and augmenting $(D - cx^1)/d(t)$ units of flow from node s to node t .

The capacity scaling algorithm performs a number of scaling phases, and in a scaling phase converts a 2Δ -optimal solution of the 2Δ -scaled problem into a Δ -optimal solution of the Δ -scaled problem. The algorithm starts with $\Delta := 2^{\lfloor \log D \rfloor}$. Notice that this value of Δ satisfies $D/2 \leq \Delta \leq D$, and D is an upper bound on the maximum flow that can be sent from node s to node t (from the nonnegative cost assumption). The algorithm converts a 2Δ -optimal solution into a Δ -optimal solution in two subphases. In the first subphase, by using the procedure called *restore-feasibility*, the algorithm converts the terminal solution of the 2Δ -scaled problem into a dual feasible solution (i.e., satisfying the optimality conditions (3)) of the Δ -scaled problem. In the second subphase, by using the procedure called *restore-optimality*, the algorithm converts this dual feasible solution into a Δ -optimal solution of the Δ -scaled problem. At the end of the last scaling phase, $\Delta = 1$, and the algorithm obtains an integer optimal solution. Next, the algorithm augments a fractional flow along the shortest path from node s to node t to obtain a real-valued optimal solution of the constrained maximum flow problem. Figures 2 and 3 give the algorithmic description of the capacity scaling algorithm for the constrained maximum flow problem, which is followed by its explanation and analysis.

```

algorithm scaling;
begin
   $x := 0; \pi := 0; v := 0;$ 
   $\Delta := 2^{\lfloor \log D \rfloor};$ 
  while  $\Delta \geq 1$  do
    begin
      { $\Delta$ -scaling phase begins here}
      restore feasibility;
      restore-optimality;
       $\Delta := \Delta/2;$ 
      { $\Delta$ -scaling phase ends here}
    end;
    determine the shortest path distances  $d(\cdot)$  from node  $s$  and update  $\pi := \pi - d$ ;
    augment  $\delta := (D - cx)/(-\pi(t))$  units of flow along the shortest path
      from node  $s$  to node  $t$ ;
    update  $x$ ,  $cx$ , and  $v$ ;
  end;

```

Figure 2. The capacity scaling algorithm for the constrained maximum flow problem.

```

procedure restore-feasibility;
begin
  for every arc in the  $\Delta$ -residual network  $G(x, \Delta)$  do
    if  $r_{ij} > 0$  and  $c_{ij}^\pi < 0$  then send  $\Delta$  units of flow along  $(i, j)$ , update  $x$ , and
      the imbalances  $e(i)$  and  $e(j)$ ;
    while there is an imbalanced node do
      begin
        select a node  $k$  with  $e(k) > 0$  and a node  $l$  with  $e(l) < 0$ ;
        determine shortest path distances  $d(\cdot)$  from node  $k$  with respect to the arc
          lengths  $c_{ij}^\pi$  and update  $\pi := \pi - d$ ;
        augment  $\Delta$  units of flow along the shortest path  $P$  from
          node  $k$  to node  $l$  in  $G(x, \Delta)$ ;
        update  $x$ ,  $cx$ ,  $e(\cdot)$ , and  $G(x, \Delta)$ ;
      end;
    end;
end;

procedure restore-optimality;
begin
  determine shortest path distances  $d(\cdot)$  from node  $s$  and update  $\pi := \pi - d$ ;
  while  $(D - cx) \geq (-\pi(t))\Delta$  do
    begin
      augment  $\Delta$  units of flow along a shortest path from node  $s$  to node  $t$ ;
      update  $x$ ,  $v$ , and  $G(x, \Delta)$ ;
      determine shortest path distance  $d(\cdot)$  from node  $s$  with respect to the arc
        lengths  $c_{ij}^\pi$  and update  $\pi := \pi - d$ ;
    end;
  end;

```

Figure 3. Procedures of the capacity scaling algorithm.

We now explain various steps of the capacity scaling algorithm. First, we take a detailed look of the procedure *improve-feasibility*. Let $x^*(2\Delta)$ denote the flow at the end of the 2Δ -scaling phase and $v^*(2\Delta)$ denote its value. When we go from the 2Δ -scaled problem to the Δ -scaled problem, the capacities of all arcs increase by 0 or Δ units. Consequently, the residual capacities too increase by 0 or Δ units. As a result, the Δ -residual network $G(x, \Delta)$ may contain some new arcs that were not present in $G(x, 2\Delta)$. If for any such arc (i, j) , $c_{ij}^\pi < 0$, then it violates the optimality condition. (Notice that all other arcs continue to satisfy the optimality condition.) We restore the optimality condition of this arc (i, j) by sending Δ units of flow on it so that it gets saturated in $G(x, \Delta)$ and drops out of the residual network $G(x, \Delta)$. This operation might add the reversal arc (j, i) to the residual network, but since $c_{ij}^\pi < 0$, we have $c_{ji}^\pi = -c_{ij}^\pi > 0$, and the reversal satisfies the optimality conditions. This explains the preprocessing step we perform at the beginning of the procedure *restore-optimality*.

Saturating some arcs of the residual network, however, creates imbalances at some nodes. This solution is a dual feasible pseudoflow (i.e., satisfies the optimality conditions) but possibly violates primal feasibility. We restore its primal feasibility by performing shortest

path augmentations. We augment Δ units of flow from excess nodes to deficit nodes along shortest paths. The strong connectivity assumption implies that we can send flow from any excess node to any deficit node. The residual capacities are always an integral multiple of Δ as may be proved via induction on the number of steps performed by the algorithm, and this allows Δ units of flow to be sent along the shortest paths. These shortest path augmentations preserve the dual feasibility of the solution (see Lemma 1) and gradually reduce the imbalances at nodes. Eventually, all nodes become balanced and the procedure terminates. Let $x^0(\Delta)$ denote the flow at this point, and $v^0(\Delta)$ denote the value of this flow. Notice that $v^0(\Delta) = v^*(2\Delta)$, because this procedure does not send any additional flow from the source node or into the sink node.

Let us now study the impact of the shortest path augmentations on the cost of flow. It follows from Theorem 1 that $x^*(2\Delta)$ is an optimal solution of the minimum cost flow problem in the 2Δ -scaled problem with flow value equal to $v^*(2\Delta)$. Next, observe that $x^*(2\Delta)$ is a feasible solution of the minimum cost flow problem in the Δ -scaled problem with flow value equal to $v^*(2\Delta)$, because $u_{ij}(\Delta) \geq u_{ij}(2\Delta)$ for every arc $(i, j) \in A$. Further, since $x^0(\Delta)$ satisfies the optimality condition (3), it is an optimal solution of the minimum cost flow problem with flow value equal to $v^0(\Delta) = v^*(2\Delta)$. The preceding two observations imply that $cx^0(\Delta) \leq cx^*(2\Delta)$.

Alternatively, the shortest path augmentations maintain the flow value but may decrease the cost of flow (because some arc capacities increase and we optimize over a larger set of feasible solutions). As the cost of flow may decrease, we may send additional flow from node s to node t and still satisfy the budget constraint of D units on the cost of flow. The procedure restore-optimality accomplishes this task by sending Δ units of flow from node s to node t along shortest paths as long as it is permitted by the budget constraint. The algorithm repeatedly determines shortest path distances $d(\cdot)$ from node s , updates $\pi := \pi - d$, and augments Δ units of flow from node s to node t along a shortest path. Recall from Section 3 that $-\pi(t)$ is the minimum cost of sending one unit of flow from node s to node t ; hence as long as $(D - cx) \geq (-\pi(t))\Delta$, we keep augmenting flows along shortest paths. When $(D - cx) < (-\pi(t))\Delta$, then the solution is Δ -optimal and the Δ -scaling phase terminates.

We now discuss the worst-case complexity of the capacity scaling algorithm. We will show that the capacity scaling algorithm performs $O(\log D)$ scaling phases, $O(m)$ shortest path augmentations in each scaling phase and, consequently, runs in $O(m \log D S(n, m))$ time. It is easy to see that the capacity scaling algorithm performs $O(\log D)$ scaling phases. The algorithm starts with $\Delta = 2^{\lfloor \log D \rfloor}$ and in each scaling phase it reduces Δ by a factor of 2. After $1 + \lfloor \log D \rfloor$ scaling phases, $\Delta = 1$, and the algorithm terminates at the end of this scaling phase. Clearly, the bottleneck operation in a scaling phase is the shortest path augmentations the algorithm performs in the procedures restore-feasibility and restore-optimality. We now focus on the number of shortest path augmentations performed by these two procedures.

The procedure restore-feasibility saturates some arcs at the beginning of the procedure by sending Δ units of flow on them. As a result of these saturations, we create excess and deficit nodes. As we saturate at most m arcs, the total excess created at all the nodes is at most $m\Delta$. Each subsequent shortest path augmentation reduces the amount of excess at some node by Δ units (as this augmentation carries Δ units). Consequently, this procedure will perform at most m shortest path augmentations. Notice, however, that in the first scaling phase, the procedure restore-feasibility will not saturate any arc and therefore no such augmentation will be performed.

We next consider the shortest path augmentations performed by the procedure restore-optimality. In the first scaling phase, each shortest path augmentation sends $\Delta \geq D/2$ units of flow and there can be at most two such augmentations. We now focus on the augmentations performed in scaling phases other than the first scaling phase. The procedure restore-feasibility obtains a feasible solution $x^0(\Delta)$ of value $v^0(\Delta)$ for the Δ -scaled problem, which may not be Δ -optimal. The procedure restore-optimality converts this solution into a Δ -optimal solution $x^*(\Delta)$ of value $v^*(\Delta)$ by performing shortest path augmentations from node s to node t , each carrying Δ units. We now show that $v^*(\Delta) \leq v^*(2\Delta) + m\Delta$, which would immediately imply that the procedure restore-optimality would perform at most m shortest path augmentations because $v^0(\Delta) = v^*(2\Delta)$. This result is the subject of our next lemma.

Lemma 3. $v^*(\Delta) \leq v^*(2\Delta) + m\Delta$.

Proof. In the Δ -scaled problem, some arc capacities are Δ units higher than the corresponding arc capacities in the 2Δ -scaled problem. Suppose, for simplicity, that in the Δ -scaled problem the capacity of *only* one arc, say (k, l) , is Δ units higher, and all other arc capacities are the same as in the 2Δ -scaled problem. We claim that in this case, the constrained maximum flow value will increase by at most Δ units. Suppose that the claim is not true and $v^*(\Delta) \geq v^*(2\Delta) + 2\Delta$. We assume that $x_{kl}^*(\Delta) = u_{ij}(\Delta) > u_{ij}(2\Delta)$, since otherwise $x^*(\Delta)$ is feasible to the 2Δ -scaled problem. Let us consider a flow decomposition of $x^*(\Delta)$, where the flow is expressed as flows along paths carrying Δ units. There must be some path that passes through the arc (k, l) . Eliminating flow on this path yields a flow, say x' , that is feasible to the 2Δ -scaled problem (because the resulting flow on arc (k, l) does not use the additional capacity) and has a flow value equal to $v^*(\Delta) - \Delta > v^*(2\Delta)$. This contradicts the optimality of the flow $x^*(2\Delta)$ for the 2Δ -scaled problem. We have thus established that if the capacity of exactly one arc increases by Δ units, then the constrained maximum flow value increases by at most Δ units. In case the capacity of each of the m arcs increases by at most Δ units, then we can apply the preceding argument inductively to show that the constrained maximum flow value increases by at most $m\Delta$ units. This establishes the lemma. $\diamond$

This lemma implies that the procedure restore-optimality performs $O(m)$ shortest path augmentations. The procedure restore-feasibility has already been shown to perform $O(m)$ shortest augmentations. As the capacity scaling algorithm executes these procedures $O(\log D)$ times, it performs $O(m \log D)$ shortest path augmentations and runs in $O(m \log D S(n, m))$ time. Hence the following theorem.

Theorem 2. *The capacity scaling algorithm obtains an optimal real (or, integer) solution of the constrained maximum flow problem with integer arc capacities in $O(m \log D S(n, m))$ time, where $S(n, m)$ is the time needed to solve a shortest path problem with nonnegative arc lengths.* ♦

Lastly, we indicate how can we satisfy the nonnegative cost assumption that we stated in Section 1. To satisfy the assumption, we execute the following procedure:

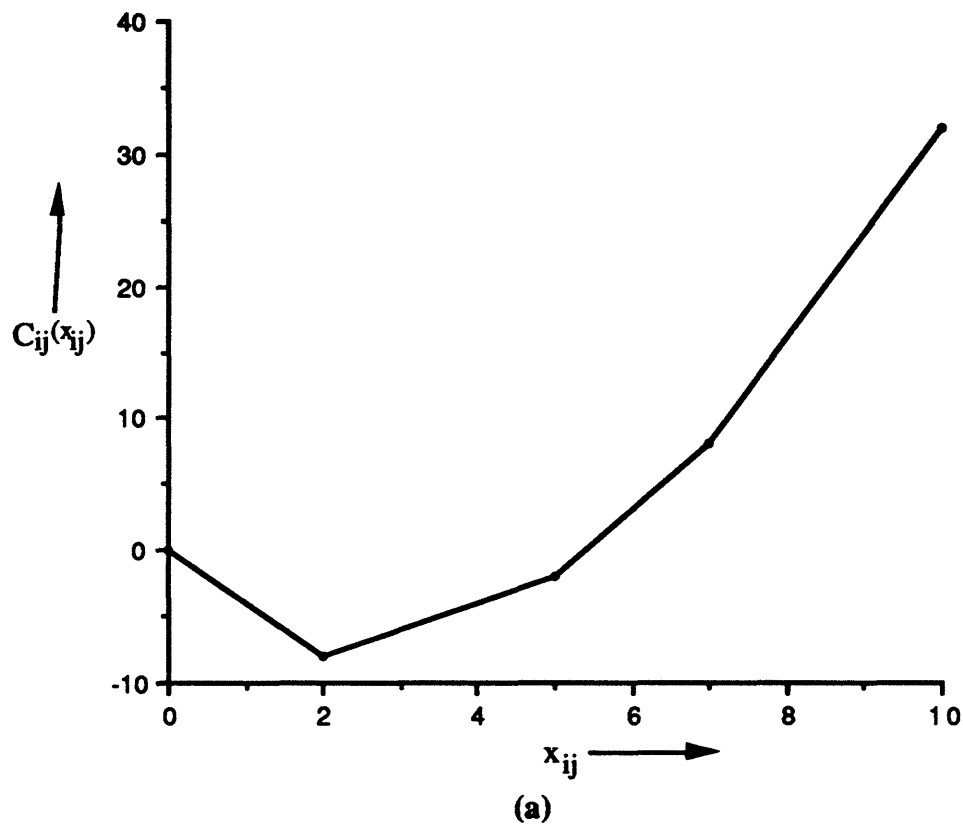
Step 1. Add an uncapacitated arc (t, s) with zero cost to the network and solve the minimum cost circulation problem in the network (i.e., the minimum cost flow problem with the supply/demand of each node equal to zero). If the optimal solution is unbounded, then the constrained maximum flow problem is also unbounded, and we stop. Otherwise, let x^* be the optimal flow and π^* be the optimal node potentials. Redefine arc costs as $c'_{ij} = c_{ij} - \pi^*(i) + \pi^*(j) \geq 0$ for each arc $(i, j) \in A$, $D' = D + |cx^*|$, and go to Step 3.

Step 2. Let G' be a subgraph of G for which $c'_{ij} = 0$ for each arc (i, j) . Starting with the flow x^* , solve a maximum flow problem from node s to node t in G' , and send this flow on arc (t, s) so that x' is a circulation. Let x' denote the resulting flow in the original network.

We now consider the residual network $G(x')$ with arc costs c' . It can be easily verified that $c' \geq 0$, and each directed path from node s to node t in $G(x')$ has a positive length. As all arc lengths are integer, each directed path from node s to node t will have length at least one. We now solve the constrained maximum flow problem with the available budget equal to D' . The running time of the constrained maximum flow problem depends on the maximum possible value of D' which we study next. The flow x' is a circulation and it follows from the flow decomposition theory that it can be decomposed into at most m cycle flows each of which saturates at least one finite capacity arc (see, for example, Ahuja, Magnanti, and Orlin [1993]). As there are at most m finite capacity arcs each with capacity at most U , and the minimum possible cost of a cycle is $-nC$, we obtain $cx' \geq -nmCU$. Consequently, $D' \leq D + nmCU$. Notice that $\log D' \leq \log D + \log n + \log m + \log C + \log U = O(\log M)$, where M is the largest single element in the data. Therefore, the running time of our capacity scaling algorithm for the constrained maximum flow problem discussed in this section becomes $O(m \log M S(n, m))$.

5. Convex Cost Flows

In this section, we study the constrained maximum flow problem in a network where the cost of flow on any arc $(i, j) \in A$ is given by a convex function $C_{ij}(x_{ij})$. The cost function $C_{ij}(x_{ij})$ may be a piecewise linear convex function (as shown in Figure 4(a)); or a continuous function stated concisely (as shown in Figure 4(b)).



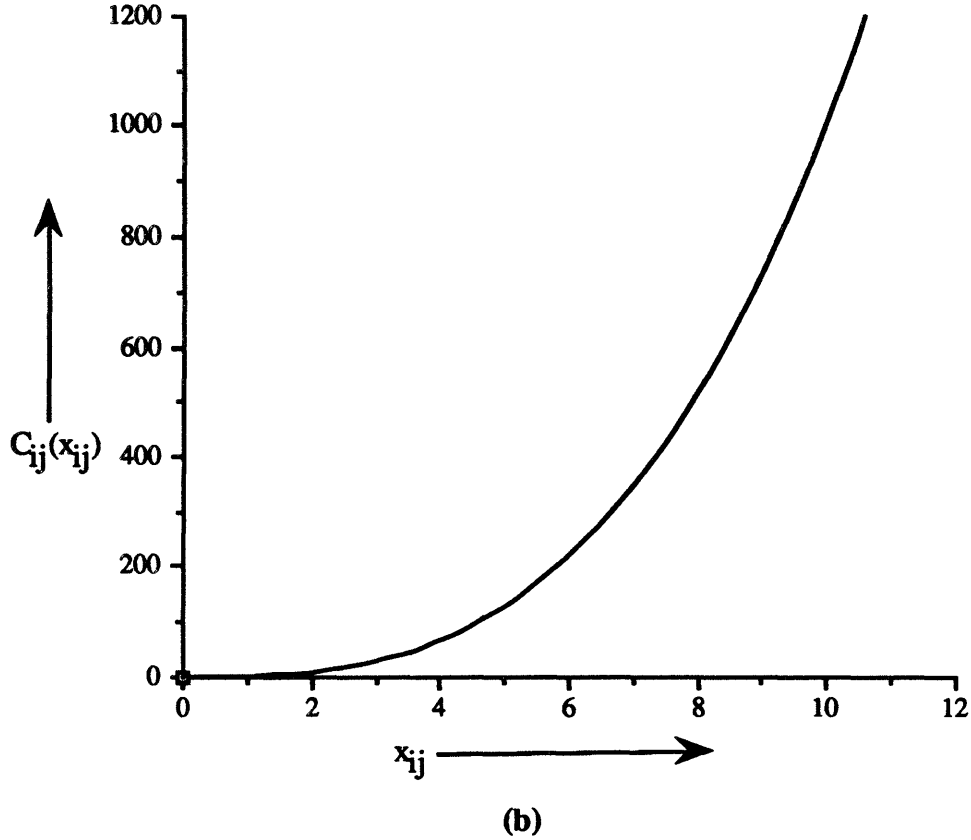


Figure 4. Two examples of convex cost functions.

We consider the convex flow problem subject to the following assumptions:

1. The cost function $C_{ij}(x_{ij})$ is linear between successive integers. (This ensures that there is an optimal solution which is integral.)
2. Each arc (i, j) has a finite capacity u_{ij} .
3. The network does not contain any negative cost cycle. (Note that we can satisfy this assumption using a method similar to the one described in Section 4, where we solve a minimum cost flow problem with convex costs.)

In this section, we generalize the capacity scaling algorithm described in the previous section to obtain a polynomial-time algorithm for the constrained maximum flow problem in convex cost networks. Our algorithm is a modification of the capacity scaling algorithm for the convex cost flow problem described in Chapter 14 of Ahuja, Magnanti and Orlin [1993], which in turn is a variant of a scaling algorithm due to Minoux [1984, 1986].

The capacity scaling algorithm for the convex cost flow problem solves a sequence of Δ -scaled problems for decreasing values of Δ . Initially $\Delta = 2^{\lfloor \log U \rfloor}$, and in each subsequent scaling phase, Δ decreases by a factor of 2. For the Δ -scaled problem, we define the arc cost function

$C_{ij}(x_{ij})$ in the following manner: $C_{ij}^\Delta(x_{ij}) = C_{ij}(x_{ij})$ whenever x_{ij} is an integer multiple of Δ , and $C_{ij}^\Delta(x_{ij})$ is linear between multiples of Δ . Consider, for example, the function $C_{ij}(x_{ij}) = x_{ij}^3$ for $0 \leq x_{ij} \leq 12$ and $C_{ij}(x_{ij}) = \infty$ for $x_{ij} \geq 12$. In the first scaling phase, the algorithm linearizes the function into segments of length 8, in the second scaling phase, the algorithm linearizes the function into segments of length 4, and so on until the segment lengths become 1. Figure 5 shows the linearizations of the function in Figure 4(b) for $\Delta = 2$.

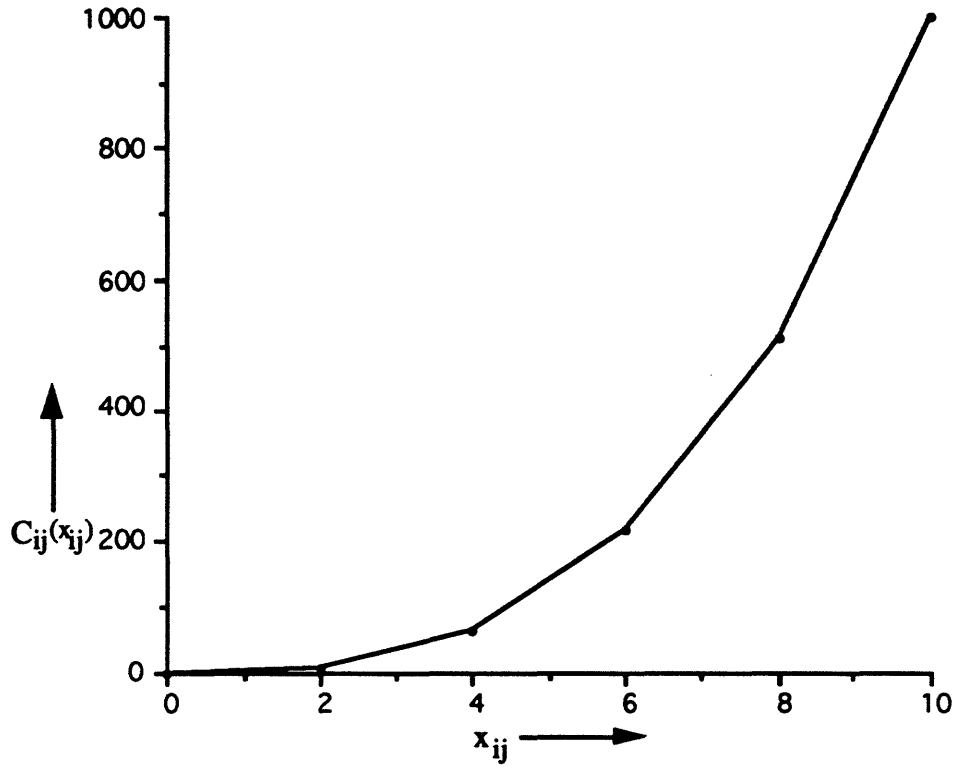


Figure 5. Linearizations of a convex function.

The Δ -scaled problem for the convex cost flow problem differs from the Δ -scaled problem for the minimum cost flow problem in the sense that the cost of flow on each arc is a piecewise linear convex function instead of a linear function. We now use a well-known result that a mathematical programming problem with piecewise linear convex cost functions and linear constraints can be transformed to a linear programming problem by introducing a separate variable for each linear segment (see, e.g., Murty [1976]). This result implies that the Δ -scaled problem for the convex cost case can be transformed into the Δ -scaled problem for the linear cost case problem by introducing a separate arc for each linear segment. For instance, consider the cost function of arc (i, j) for the 2-scaled problem shown in Figure 5, for which the transformed minimum cost flow problem will contain 5 arcs with different arc costs. We refer to these arcs as $(i, j)^1, (i, j)^2, \dots, (i, j)^5$.

An advantage of this transformation is that it transforms a Δ -scaled problem for the convex cost case into a Δ -scaled problem for the linear cost case, and thereby allows one to use the approach discussed in the previous section. However, a drawback of this transformation is that it expands the size of the network (i.e., the number of arcs) substantially. We can overcome this drawback by not actually expanding the network and treating the additional arcs implicitly. We now discuss how we can do that. Consider the residual network corresponding to the Δ -scaled problem for the transformed minimum cost flow problem. For this purpose, we focus on a single arc (i, j) of the original network with multiple copies in the transformed network. Suppose that $x_{ij} = 4$ in the original network, which translates into 2 units of flow on each of the arcs $(i, j)^1$ and $(i, j)^2$, and zero flow on the rest of the arcs of the transformed network. Consequently, the residual network contains the arcs $(i, j)^3, \dots, (i, j)^5$, and the reversals of the arcs $(i, j)^1$ and $(i, j)^2$ (which we denote by $(j, i)^1$ and $(j, i)^2$). Now observe that if we have to send flow from node i to node j , we will send it using the arc $(i, j)^3$ (because it is cheapest). In case we have to send flow from node j to node i , then we will send it using the arc $(j, i)^2$. This observation implies that in the Δ -residual network we need not maintain multiple copies between this node pair; maintaining just the two arcs, $(i, j)^3$ and $(j, i)^2$, is sufficient because these are the arcs that matter at this point. The preceding discussion suggests the following method to construct the Δ -residual network in the Δ -scaling phase: For each arc $(i, j) \in A$, the Δ -residual network contains the arc (i, j) with the residual capacity Δ and a unit cost equal to $(C_{ij}(x_{ij} + \Delta) - C_{ij}(x_{ij}))/\Delta$. Further, for each arc $(i, j) \in A$ with $x_{ij} \geq \Delta$, the Δ -residual network contains the arc (j, i) with a residual capacity Δ and a unit cost equal to $(C_{ij}(x_{ij} - \Delta) - C_{ij}(x_{ij}))/\Delta$.

We are now in a position to describe our algorithm for the constrained maximum flow problem in convex cost networks. Initially, $\Delta = 2^{\lfloor \log U \rfloor}$ and we initialize the algorithm with the zero pseudoflow x and zero node potential π . The algorithm then solves a sequence of Δ -scaled problems with decreasing values of Δ and obtains Δ -optimal solutions, until $\Delta = 1$, when it terminates. We next describe how the algorithm transforms a 2Δ -optimal solution into a Δ -optimal solution. We begin the Δ -scaling phase when the 2Δ -scaling phase terminates. In the 2Δ -scaling phase, we linearize $C_{ij}(x_{ij})$ by segments of length 2Δ , and in the Δ -scaling phase we linearize this cost function by segments of length Δ . Consequently, the arc costs change. As a result, the reduced costs of the arcs also change and the new values might become negative. As shown in Ahuja, Magnanti and Orlin [1993], one can then adjust flow on each arc (i, j) by at most Δ units so as to make the reduced costs of both the arcs, (i, j) and (j, i) , in the Δ -residual network nonnegative.

The preceding discussion shows that by sending Δ units of flow on at most m arcs, we can obtain a flow in the transformed network that satisfies the optimality conditions. This, however, creates excesses and deficits at nodes. We then execute the procedure restore-

feasibility which converts the pseudoflow into a flow within m shortest path augmentations. We next execute the procedure restore-optimality which augments flow from node s to node t along shortest paths as long as the cost of flow is no greater than D . The correctness arguments we gave in Section 4 also hold for the convex cost case because we are solving the transformed problem which has linear costs. It can be easily verified that the result of Lemma 3 holds for the convex case too and the procedure restore-optimality too performs at most m shortest path augmentations. We have thus established the following theorem.

Theorem 3. *The capacity scaling algorithm obtains an integer optimal flow for the constrained maximum flow problem with convex costs in $O(m \log U S(n, m))$ time. ♦*

ACKNOWLEDGEMENTS

This research was supported by the Air Force Office of Scientific Research Grant AFORS-88-0088 as well as grants from UPS and Prime Computer Corporation.

REFERENCES

- Ahuja, R. K., T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, New Jersey.
- Ahuja, R. K., A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. 1992. Finding minimum cost flows by double scaling. *Mathematical Programming* 53, 243-266.
- Ahuja, R. K., K. Mehlhorn, J. B. Orlin, and R. E. Tarjan. 1990. Faster algorithms for the shortest path problem. *Journal of ACM* 37, 213-223.
- Edmonds, J., and R. M. Karp. 1972. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of ACM* 19, 248-264.
- Fredman, M. L., and R. E. Tarjan. 1984. Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 338-346. Full paper in *Journal of ACM* 34 (1987), 596-615.
- Fulkerson, D. R., 1959. Increasing the capacity of a network: The parametric budget problem. *Management Science* 5, 473-483.
- Goldberg, A. V., and R. E. Tarjan. 1987. Solving minimum cost flow problem by successive approximation. *Proceedings of the 19th ACM Symposium on the Theory of Computing*, pp 7-18. Full paper in *Mathematics of Operations Research* 15 (1990), 430-466.
- Johnson, D. B. 1982. A priority queue in which initialization and queue operations take $O(\log \log D)$ time. *Mathematical Systems Theory* 15, 295-309.

- Minoux, M. 1984. A polynomial algorithm for minimum quadratic cost flow problems. *European Journal of Operational Research* **18**, 377-387.
- Minoux, M. 1986. Solving integer minimum cost flows with separable convex cost objective polynomially. *Mathematical Programming Study* **26**, 237-239.
- Murty, K. G. 1976. *Linear and Combinatorial Programming*. John Wiley & Sons.
- Orlin, J. B. 1988. A faster strongly polynomial minimum cost flow algorithm. *Proceedings of the 20th ACM Symposium on the Theory of Computing*, pp. 377-387. Full paper in *Operations Research* **41**(1993) 338-350.