Journal of Global Optimization **19:** 121–139, 2001.
© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

Linear Approximations in a Dynamic Programming Approach for the Uncapacitated Single-Source Minimum Concave Cost Network Flow Problem in Acyclic Networks [⋆]

RAINER E. BURKARD[1], HELIDON DOLLANI[1] and PHAN THIEN THACH[2]
[1]*Technische Universität Graz, Institut für Mathematik, Steyrergasse 30, A-8010 Graz, Austria (e-mail: {burkard, dollani}@opt.math.tu-graz.ac.at.);* [2]*Institute of Mathematics, P.O. Box 631, Bo Ho, Hanoi, Vietnam.*

(Received 7 February 2000; accepted in revised form 21 September 2000)

Abstract. We consider minimum concave cost flow problems in acyclic, uncapacitated networks with a single source. For these problems a dynamic programming scheme is developed. It is shown that the concave cost functions on the arcs can be approximated by linear functions. Thus the considered problem can be solved by a series of linear programs. This approximation method, whose convergence is shown, works particularly well, if the nodes of the network have small degrees. Computational results on several classes of networks are reported.

Key words: Uncapacitated single-source acyclic networks, Concave costs, Dynamic programming, Linear approximation, Convergence

1. Introduction

We consider a directed, acyclic graph $G = (V, E)$ with vertex set $V = \{v_0, v_1, \dots, v_n\}$ and arc set E. For each vertex we denote by $E_s(v)$ and $E_t(v)$, respectively, the set of arcs which start and end, resp., in v. A vertex with empty set $E_t(v)$ is called a *source*. Since G is acyclic we can assume that every arc has the form (v_i, v_j) with $i < j$. We furthermore assume that v_0 is the single source in G. For any vertex v_i $(1 \leqslant i \leqslant n)$ let the nonnegative real number $q(v_i)$ be the *demand* of vertex v_i. The total demand of the network is given by $Q = \sum_{i=1}^{n} q(v_i)$.

A flow x in the network is a function $x : E \to \mathbb{R}_+$ which fulfills the flow conservation conditions

$$\sum_{e \in E_t(v_i)} x(e) - \sum_{e \in E_s(v_i)} x(e) = q(v_i) \quad \text{for } i = 1, 2, \dots, n \tag{1}$$

[⋆] This research has been supported by Spezialforschungsbereich F003 ‘Optimierung und Kontrolle’, Projektbereich Diskrete Optimierung.

Now we define on each arc $e \in E$ a concave, nonnegative cost function f_e. For example, f_e might be a fixed charge function of the form

$$f_e(x(e)) = \begin{cases} 0 & \text{if } x(e) = 0 \\ \alpha x(e) + \beta & \text{if } x(e) > 0. \end{cases} \tag{2}$$

where α and β might depend on the arc e. Note that we do not have to consider parallel arcs, since the sum of concave functions is concave again. This means that any arc e is uniquely defined by its head and tail.

In this paper we design a solution method for the minimum concave cost network flow problem (P) in uncapacitated, acyclic networks with a single source.

(P) Find a flow $x(e), e \in E$, which minimizes the total cost

$$\sum_{e \in E} f_e(x)$$

The existence of an optimal solution of (P) follows from the nonnegativity of the cost function $f_e(x)$.

The minimum concave cost network flow problem arises in many application areas, see e.g. [5] and the references therein. In the case of fixed charge costs the problem can be reformulated as a mixed integer linear program and can be solved by branch and bound methods. Gallo, Sandi and Sodini [4] present an efficient branch and bound method which can be applied to problems with general concave functions and which takes also capacity constraints into account. Guisewite and Pardalos [6] discuss algorithms, applications and complexity issues for the uncapacitated single-source minimum concave cost network flow problem.

The best exact methods known for general minimum concave cost network flow problems work only for small to medium sized problem instances. To solve large scale problems, several authors have proposed heuristics. Recent work in this direction includes the GRASP heuristic of Holmqvist, Migdalas and Pardalos [7] for the single source uncapacitated minimum concave cost network flow problem and the dynamic slope scaling procedure for the minimum concave cost network flow problem with cost functions which are either of fixed-charge type or are concave and piecewise linear by Kim and Pardalos [8, 9].

Another approach is to consider special cases of the minimum cost network flow problem which arise if additional structure is imposed on the input data. Klinz and Tuy [10] and Tuy et al. [12] consider problems where nonlinear concave cost functions are imposed on a restricted number of arcs. They design strongly polynomial algorithms for this special case. For a survey on the minimum concave cost network flow problem with a fixed number of nonlinear arc costs see the recent paper by Tuy [11]. Guisewite and Pardalos [5] discuss a large number of applications, complexity results and solution methods for the minimum concave cost network flow problem.

In this paper we present a dynamic programming approach for the minimum concave cost network flow problem in uncapacitated, acyclic networks with a single source which is based on linear approximations for the cost functions. This approach seems to be well suited for layered type networks and graphs with small vertex degrees.

The paper is organized as follows: in Section 2 we outline a general dynamic programming scheme for solving (P). In Section 3 we introduce linear approximations. In Section 4 the convergence of the approximation method is shown. In Section 5 we report on computational results on problem instances of various structures and close finally the paper with concluding remarks.

2. A general dynamic programming scheme

For $\ell = 0, 1, \dots, n$ let G_ℓ be the subgraph of G induced by the vertex set $V_\ell = \{v_0, v_1, \dots, v_\ell\}$. Let $U_\ell := V \backslash V_\ell$ and define $C_\ell \subseteq V_\ell$ as the set of all vertices from which there leads an arc to a vertex in U_ℓ. We can assume $|C_\ell| \geqslant 2 \;\; (1 \leqslant l \leqslant n-1)$, since otherwise we could decompose the given problem into two independent subproblems.

For numerical reasons we assume that $|C_\ell|$ is not too large, since the amount of work proposed by the following dynamic programming scheme depends on this size.

In the following we present a general dynamic programming method for problem (P). We consider n stages corresponding to $v_1, v_2, \dots, v_n$, respectively. The vector of state variables in stage ℓ is a vector x_ℓ defined on $E_t(v_\ell)$:

$$x_\ell = (x(e)), \; e \in E_t(v_\ell).$$

For every $\ell = 0, 1, \dots, n-1$ we define

$$z_\ell := (x_{\ell+1}, x_{\ell+2}, \dots, x_n).$$

According to this notation, $z_{n-1}{=}x_n$, $z_{\ell-1}{=}(x_\ell, z_\ell)$, $\ell = 1, 2, \dots, n-1$ and $z_0{=}(x_1, x_2, \dots, x_n)$. Given a nonnegative vector z_ℓ, $1 \leqslant \ell \leqslant n-1$, we define a vector q_{z_ℓ} on the set of vertices $\{v_1, v_2, \dots, v_\ell\}$ as follows

$$q_{z_\ell}(v) = \begin{cases} q(v) & \text{if } v \notin C_\ell \\ q(v) + \sum\limits_{(v,u)\in E} z_\ell(v,u) & \text{if } v \in C_\ell. \end{cases} \tag{3}$$

For $\ell = 1, 2, \dots, n-1$ we denote by $h_\ell(z_\ell)$ the minimum cost value on the set of feasible flows in G_ℓ with respect to the demand vector q_{z_ℓ}. In addition, we define $h_0(z_0) \equiv 0$.

Since G_ℓ can be obtained by adding vertex v_ℓ and the set of arcs $E_t(v_\ell)$ to $G_{\ell-1}$, we have the following recursive equations

$$h_\ell(z_\ell) = \min_{x_\ell} \left\{ h_{\ell-1}(x_\ell, z_\ell) + \sum_{e\in E_t(v_\ell)} f_e(x_\ell(e)) \,\middle|\, \sum_{e\in E_t(v_\ell)} x_\ell(e) = q_{z_\ell}(v_\ell),\ x_\ell \geqslant 0 \right\},$$

$\ell = 1, 2, \dots, n-1$. On the basis of these recursive equations we can, in principle, find an optimal solution to problem (P) by the following backward scheme:

Subproblem in Stage n:

$$\begin{aligned} \min\ & h_{n-1}(x_n) + \sum_{e\in E_t(v_n)} f_e(x_n(e)), \\ \text{s.t.}\ & \sum_{e\in E_t(v_n)} x_n(e) = q(v_n),\ x_n \geqslant 0. \end{aligned}$$

Denote by $\overline{\gamma}_n$ and $\overline{x}_n$ the optimal value and optimal solution to this problem, respectively, and define $\overline{z}_{n-1} = \overline{x}_n$.

Subproblem in Stage $n-1$:

$$\begin{aligned} \min\ & h_{n-2}(x_{n-1}, \overline{z}_{n-1}) + \sum_{e\in E_t(v_{n-1})} f_e(x_{n-1}(e)), \\ \text{s.t.}\ & \sum_{e\in E_t(v_{n-1})} x_{n-1}(e) = q_{\overline{z}_{n-1}}(v_{n-1}),\ x_{n-1} \geqslant 0. \end{aligned}$$

Denote by $\overline{\gamma}_{n-1}$ and $\overline{x}_{n-1}$ the optimal value and optimal solution to this problem, respectively, and define $\overline{z}_{n-1} = (\overline{x}_{n-1}, \overline{x}_n)$.

Analogously we obtain for the subproblem in Stage ℓ:

$$\begin{aligned} \min\ & h_{\ell-1}(x_\ell, \overline{z}_\ell) + \sum_{e\in E_t(v_\ell)} f_e(x_\ell(e)), \\ \text{s.t.}\ & \sum_{e\in E_t(v_\ell)} x_\ell(e) = q_{\overline{z}_\ell}(v_\ell),\ x_\ell \geqslant 0. \end{aligned}$$

Denote by $\overline{\gamma}_\ell$ and $\overline{x}_\ell$ the optimal value and optimal solution to this problem, respectively, and define $\overline{z}_{\ell-1} = (\overline{x}_\ell, \overline{z}_\ell)$.

...

Subproblem in Stage 1:

$$\begin{aligned} \min\ & \sum_{e\in E_t(v_1)} f_e(x_1(e)), \\ \text{s.t.}\ & \sum_{e\in E_t(v_1)} x_1(e) = q_{\overline{z}_1}(v_1),\ x_1 \geqslant 0. \end{aligned}$$

(Since $E_t(v_1)$ contains only a single arc (v_0, v_1), this problem has the feasible solution $x_1 = q_{\bar{z}_1}(v_1)$.) As usual we denote by $\overline{\gamma}_1$ and $\overline{x}_1$ the optimal value and optimal solution of the above problem, and define $\overline{z}_0 = (\overline{x}_1, \overline{z}_1)$.
Now we get:

THEOREM 2.1. *By solving the subproblems in n stages we obtain an optimal flow $\overline{z}_0 = (\overline{x}_1, \overline{x}_2, \dots, \overline{x}_n)$ and its optimal value $\overline{\gamma}_n$.*

Proof. Since by adding vertex v_n and the set of arcs $E_t(v_n)$ to G_{n-1} we obtain the given graph G, the value $\overline{\gamma}_n$ given in the subproblem in stage n is the optimal value to our problem. Since $\overline{x}_\ell$ solves the subproblem in stage ℓ, it follows from the recursive equations that

$$\begin{aligned}
\overline{\gamma}_n &= \sum_{e \in E_t(v_n)} f_e(\overline{x}_n(e)) + h_{n-1}(\overline{z}_{n-1}) \\
&= \sum_{e \in E_t(v_n)} f_e(\overline{x}_n(e)) + \sum_{e \in E_t(v_{n-1})} f_e(\overline{x}_{n-1}(e)) + h_{n-2}(\overline{z}_{n-2}) \\
&\dots \\
&= \sum_{e \in E_t(v_n)} f_e(\overline{x}_n(e)) + \sum_{e \in E_t(v_{n-1})} f_e(\overline{x}_{n-1}(e)) + \dots + \sum_{e \in E_t(v_1)} f_e(\overline{x}_1(e)).
\end{aligned}$$

This proves the optimality of flow $\overline{z}_0 = (\overline{x}_1, \overline{x}_2, \dots, \overline{x}_n)$. □

Since in each subproblem the objective function is concave (see Lemma 4.3) and the feasible set is a simplex, the subproblems can be solved by evaluating the objective function in the vertices of the simplex. Note, however, that the functions $h_\ell(x_\ell)$ are only implicitly given and their actual computation depends on the number of states which have to be taken into consideration. This might be very costly. Therefore we design in the next section a faster approximation method and show the convergence of the approximate solutions to the optimal solution of (P).

3. A linear approximation method

A major complication in the general method presented in Section 2 is that the functions $h_\ell(x_\ell)$ are implicitly defined and hard to compute. Therefore we shall use successive linear underestimations for these functions h_ℓ in an iterative scheme which will be shown in Section 4 to converge towards the optimal solution of (P).

In each iteration of the approximation method we perform the following tasks

- Given linear approximations of the functions h_ℓ, we solve approximate subproblems in a backward scheme. The subproblem in stage n provides a lower bound for the optimal value of (P)
- Using the solutions of the subproblems, we update the linear approximations for the functions h_ℓ, $\ell = 1, 2, \dots, n-1$.

- Using local search, based on the information about the solution of the subproblems we improve, if possible, the upper bound on the optimal value of (P).

(3) will be replaced by

$$q_{s_\ell}(v) := \begin{cases} q(v) & \text{if } v \notin C_\ell \\ q(v) + s_\ell(v) & \text{if } v \in C_\ell \end{cases}$$

where s_ℓ is a vector defined on C_ℓ, $\ell = 1, 2, \dots, n-1$. The minimum cost value on the set of feasible flows of G_ℓ with respect to the demand vector s_ℓ will be denoted by $H_\ell(s_\ell)$. For notational convenience we define $H_0(s_0) \equiv 0$ for $s_0 \geqslant 0$. Moreover, we define $H_0^k = H_0 \equiv 0$ for any $k = 1, 2, \dots$

After these preparations we can now formulate our approximation algorithm.

Linear Approximation Method.
Let ρ be a given tolerance.
Initialization. Let $\overline{z}^0$ be an initial feasible flow with cost $\delta^0 = \sum_{e\in E} f_e(\overline{z}^0(e))$.
δ^0 is an upper bound on the optimal value. Define $H_\ell^1 \equiv 0$ for $\ell = 1, 2, \dots, n-1$,
$k := 1$.
Iteration k

– **Step 1.** Solve the following approximate subproblems in the backward scheme.

  - **Approximate Subproblem in Stage n:** Let $s_{n-1}(v) = x_n(v, v_n)$, for all $v \in C_{n-1}$, and solve

$$\begin{aligned} \min \ & H_{n-1}^k(s_{n-1}) + \sum_{e\in E_t(v_n)} f_e(x_n(e)), \\ \text{s.t.} \ & \sum_{e\in E_t(v_n)} x_n(e) = q(v_n), \ x_n \geqslant 0. \end{aligned}$$

    Let γ_n^k and (x_n^k, s_{n-1}^k) be the optimal value and optimal solution of that problem, respectively.

    In general, we consider in stage ℓ the following subproblem:

  - **Approximate Subproblem in Stage ℓ:** According to $z_\ell^k = (x_{\ell+1}^k, \dots, x_n^k)$ and s_ℓ^k already obtained, define $s_{\ell-1}$ as follows

$$s_{\ell-1}(v) := \sum_{(v,u)\in E} z_\ell^k(v,u) + x_\ell(v, v_\ell), \quad v \in C_{\ell-1},$$

and solve

$$\begin{aligned}
\min\ & H^k_{\ell-1}(s_{\ell-1}) + \sum_{e\in E_t(v_\ell)} f_e(x_\ell(e)),\\
\text{s.t.}\ & \sum_{e\in E_t(v_\ell)} x_\ell(e) = q_{s^k_\ell}(v_\ell),\ x_\ell \geqslant 0.
\end{aligned}$$

Let γ^k_ℓ and $(x^k_\ell, s^k_{\ell-1})$ be the optimal value and optimal solution, respectively.

Finally we get

- **Approximate Subproblem in Stage 1:** According to $z^k_1 = (x^k_2, \dots, x^k_n)$ and s^k_1 already obtained, define s_0 as follows

$$s_0(v_0) \ = \sum_{(v,u)\in E} z^k_1(v_0, u) + x_1(v_0, v_1),$$

and solve

$$\min\ H^k_0(s_0) + \sum_{e\in E_t(v_1)} f_e(x_1(e)),$$

$$\text{s.t.}\ \sum_{e\in E_t(v_1)} x_1(e) = q_{s^k_1}(v_1),\ x_1 \geqslant 0.$$

Let γ^k_1 and x^k_1 be the optimal value and optimal solution, respectively, and define $z^k_0 = (x^k_1, x^k_2, \dots, x^k_n)$.

– **Step 2.** Improve the linear approximations: For every $\ell = 1, 2, \dots, n-1$, define the function H^{k+1}_ℓ on the domain of s_ℓ such that the set $\{(\gamma, s_\ell)|\ \gamma \leqslant H^{k+1}_\ell(s_\ell),\ s_\ell \geqslant 0\}$ is the convex hull of the vector $(\gamma^k_\ell, s^k_\ell)$ and the set $\{(\gamma, s_\ell)|\ \gamma \leqslant H^k_\ell(s_\ell),\ s_\ell \geqslant 0\}$.
– **Step 3.** (optional). Using the best flow $\overline{z}^{k-1}$ of the previous iteration and the flow $z^k_0 = (x^k_1, x^k_2, \dots, x^k_n)$, update flow $\overline{z}^k$ at this iteration by local search.
– **Step 4.** Define $\delta^k = \sum_{e\in E} f_e(\overline{z}^k(e))$ as an upper bound for the optimal value at this iteration. If

$$\delta^k \ - \ \gamma^k_n \ \leqslant \ \rho$$

then $\overline{z}^k$ is ρ-optimal, otherwise go to iteration $k+1$.

The functions $H^k_\ell,\ \ell = 1, 2, \dots, n-1;\ k = 1, 2, \dots$ are polyhedral concave on their domains by construction. Minimizing this type of concave functions subject to linear constraints is in general a hard problem, see Falk and Hoffman [2]. In our case, however, we have only one constraint and the feasible region is a simplex. Since a concave function attains its minimum in a vertex of the feasible region, we have to evaluate the functions H_ℓ^k only in the vertices of the simplex. This can be achieved by solving a linear program as follows.

Let s be a vector and let H be a function defined on the domain $\{s \mid s \geqslant 0\}$. The hypograph of H is given by

$$\text{hypo } H \ := \ \{(\gamma, s) \mid \gamma \leqslant H(s), \ s \geqslant 0\}.$$

Let *conv*(hypo H) denote the convex hull of the set hypo H.

In our case we have

$$\text{hypo } H_\ell^1 = \{(\gamma, s_\ell) \,|\gamma \leqslant 0, s_\ell \geqslant 0\} \ \ (1 \leqslant \ell \leqslant n-1)$$

and

$$\begin{aligned} \text{hypo } H_\ell^{k+1} &= \text{conv}(\text{hypo } H_\ell^k, (\gamma_\ell^k, s_\ell^k)) \\ &= \text{conv}(\text{hypo } H_\ell^1, (\gamma_\ell^1, s_\ell^1), (\gamma_\ell^2, s_\ell^2) \ldots, (\gamma_\ell^k, s_\ell^k)), \end{aligned}$$

for $k = 1, 2, \ldots$. In order to evaluate $H_\ell^{k+1}(s)$ we can solve the following linear program

$$(\alpha) \begin{cases} \max \ \displaystyle\sum_{j=1}^{k} \gamma_\ell^j r_j \\ \text{s.t.} \ \displaystyle\sum_{j=1}^{k} s_\ell^j r_j \leqslant s \\ \quad\ \ \displaystyle\sum_{j=1}^{k} r_j \leqslant 1 \\ \quad\ \ r_j \geqslant 0 \text{ for all } j = 1, 2, \ldots, k. \end{cases}$$

If C_ℓ has m_ℓ vertices, the corresponding linear program (α) has $m_\ell + 1$ constraints and k variables in the $(k+1)$st iteration. The constraints of this linear program define a simplex. At stage ℓ this simplex has $n_\ell = |E_t(v_\ell)|$ vertices. This means, the linear program (α) has to be solved n_ℓ times with the same coefficient matrix but different right hand sides. The right hand sides of two different problems at Stage l differ in only two entries. Therefore one can take advantage of this property by using the dual simplex method.

Another useful observation is that the linear programs at stage ℓ in iteration $k+1$ differ from those at iteration k only by an additional column. This can also be taken into consideration for speeding up the solution process.

Moreover, we note that the variable r_i of a linear program at stage ℓ will never enter an optimal basis at an iteration $k' \geqslant k$, if

$$\gamma_\ell^i \ \leqslant \ \max \left\{ \sum_{j \neq i} \gamma_\ell^j r_j \ \middle| \ \sum_{j \neq i} s_\ell^j r_j \leqslant s_\ell^i, \ \sum_{j \neq i} r_j \leqslant 1, \ 1 \leqslant j \leqslant k, \ j \neq i \right\}. \quad (4)$$

So we can discard redundant variables, whenever this condition is fulfilled. This results in a reduced size of the linear programs.

4. Convergence of the linear approximation method

In this section we provide a convergence proof for the method described in Section 3. In particular we prove the following theorem.

THEOREM 4.1. *(i) At any iteration k of the linear approximation method, δ^k and γ_n^k are an upper bound and a lower bound for the optimal value, respectively.*

(ii) For any $\rho \geqslant 0$, the linear approximation method terminates after a finite number of iterations yielding a ρ-optimal solution $\overline{z}^k$.

We prove this theorem by the help of the following lemmas.

LEMMA 4.2. *The minimum cost value $H(q)$ on the set of feasible flows in G with respect to the demand vector q, is a concave function of q on the domain $\{q|\, q \geqslant 0\}$.*

Proof. For any nonnegative demand vector q, the minimum cost value $H(q)$ is attained by a flow on a spanning tree in G. Let q, q' be two nonnegative vectors, and let q'' be a convex combination of q and q' : $q'' = \mu q + \mu' q'$ with $\mu + \mu' = 1$, $\mu, \mu' \geqslant 0$. Obviously, $q'' \geqslant 0$.

Let Δ be the spanning tree, for which the minimum value $H(q'')$ is attained. Every demand vector defines a feasible flow on a spanning tree in a unique way. Let x, x', and x'' be the feasible flows on Δ which correspond to q, q' and q'', respectively. Since $q'' = \mu q + \mu' q'$ we get $x'' = \mu x + \mu' x'$.

From $\sum\limits_{e\in E} f_e(x''(e)) = H(q'')$ and $\sum\limits_{e\in E} f_e(x(e)) \geqslant H(q)$, $\sum\limits_{e\in E} f_e(x'(e)) \geqslant H(q')$ we get

$$\begin{aligned}
H(q'') &= \sum_{e\in E} f_e(x''(e)) = \sum_{e\in E} f_e\left(\mu x(e) + \mu' x(e)\right) \\
&\geqslant \sum_{e\in E} \left(\mu f_e(x(e)) + \mu' f_e(x'(e))\right) \qquad \text{(since } f_e \text{ are concave)} \\
&= \mu \sum_{e\in E} f_e(x_e) + \mu' \sum_{e\in E} f_e(x'(e)) \geqslant \mu H(q) + \mu' H(q').
\end{aligned}$$

Thus, $H(q)$ is concave on $\{q \mid q \geqslant 0\}$. □

LEMMA 4.3. *For any $\ell = 1, 2, \ldots, n-1$ the functions h_ℓ and H_ℓ are concave on their domains.*

Proof. The mappings $z_\ell \rightsquigarrow q_{z_\ell}$ and $s_\ell \rightsquigarrow q_{s_\ell}$ are linear transformations from the domains $\{z|z_\ell \geqslant 0\}$ and $\{s_\ell|s_\ell \geqslant 0\}$ to the domain $\{q \mid q(v_i) \geqslant 0,\ i = 1, 2, \cdots, l\}$. By applying Lemma 4.2 to the network G_ℓ we obtain immediately the result. □

LEMMA 4.4. *For any $k = 1, 2, \dots$ and any $\ell = 0, 1, \dots, n-1$, $H_\ell^k \leqslant H_\ell$ on the domain $\{s_\ell | s_\ell \geqslant 0\}$.*

Proof. We prove this lemma by induction in k and in ℓ. First of all, since H_ℓ is nonnegative and H_ℓ^1 equals 0 on $\{s_\ell |\ s_\ell \geqslant 0\}$, it is obvious that $H_\ell^1 \leqslant H_\ell$, for all $\ell = 0, 1, \dots , n-1$. Suppose that

$$H_\ell^k \leqslant H_\ell \tag{5}$$

for all $\ell = 0, 1, \dots , n-1$. For the induction in k we are going to show that

$$H_\ell^{k+1} \leqslant H_\ell \tag{6}$$

for all $\ell = 0, 1, \dots , n-1$. For $\ell = 0$, inequality (6) is obvious, since $H_0 \equiv H_0^1 \equiv \cdots \equiv H_0^k \equiv 0$. So, we can suppose that

$$H_\ell^{k+1} \leqslant H_\ell \tag{7}$$

for some $\ell \in \{0, 1, \dots , n-1\}$. For the induction in ℓ we are going to show that

$$H_{\ell+1}^{k+1} \leqslant H_{\ell+1}. \tag{8}$$

By construction,

$$\text{hypo} H_{\ell+1}^{k+1} = \text{conv}\left(\text{hypo} H_{\ell+1}^k, (\gamma_{\ell+1}^k, s_{\ell+1}^k)\right).$$

It follows from the induction hypothesis (5) that

$$H_{\ell+1}^k \leqslant H_{\ell+1}.$$

This together with the concavity of $H_{\ell+1}$ (Lemma 4.3) implies (8) if we can show that

$$\gamma_{\ell+1}^k \leqslant H_{\ell+1}(s_{\ell+1}^k). \tag{9}$$

By definition, $\gamma_{\ell+1}^k$ is the minimum value in the approximate subproblem at stage $\ell+1$ in iteration k:

$$\gamma_{\ell+1}^k = \min\left\{ H_\ell^k(s_\ell) + \sum_{e \in E_t(v_{\ell+1})} f_e(x_{\ell+1}(e)) \,\middle|\, \sum_{e \in E_t(v_{\ell+1})} x_{\ell+1}(e) = q_{s_{\ell+1}^k}(v_{\ell+1}),\ x_{\ell+1} \geqslant 0 \right\}. \tag{10}$$

But $H_{\ell+1}\left(s_{\ell+1}^k\right)$ is by definition the minimum cost value on the set of feasible flows in $G_{\ell+1}$ with respect to the demand vector $q_{s_{\ell+1}^k}$.

It fulfills the recursion

$$H_{\ell+1}(s^k_{\ell+1}) = \min\left\{ H_\ell(s_\ell) + \sum_{e \in E_t(v_{\ell+1})} f_e(x_{\ell+1}(e)) \,\middle|\, \sum_{e \in E_t(v_{\ell+1})} x_{\ell+1}(e) = q_{s^k_{\ell+1}}(v_{\ell+1}),\ x_{\ell+1} \geqslant 0 \right\}, \tag{11}$$

where s_ℓ is defined as in the approximate subproblem in stage $\ell + 1$, iteration k. Now inequality (9) can be obtained from (10), (11) and the induction hypothesis (5). So we obtain (8). Then, inequality (6) follows by induction on ℓ, and the lemma follows by induction on k. □

Now we can prove the convergence theorem. γ^k_n denotes the optimal value of the approximate subproblem at stage n, that is

$$\gamma^k_n = \min\left\{ H^k_{n-1}(s_{n-1}) + \sum_{e \in E_t(v_n)} f_e(x_n(e)) \,\middle|\, \sum_{e \in E_t(v_n)} x_n(e) = q(v_n),\ x_n \geqslant 0 \right\}.$$

Thus we get from $H^k_{n-1} \leqslant H_{n-1}$ (Lemma 4.4) that

$$\gamma^k_n \leqslant \min\left\{ H_{n-1}(s_{n-1}) + \sum_{e \in E_t(v_n)} f_e(x_n(e)) \,\middle|\, \sum_{e \in E_t(v_n)} x_n(e) = q(v_n),\ x_n \geqslant 0 \right\}.$$

By definition of H_{n-1}, however, the minimum on the right hand side of the above inequality is the optimal value of the approximate problem in the k-th iteration. So, γ^k_n is a valid lower bound for the optimal value. If the linear approximation method terminates at iteration k, then the current best flow $\overline{z}^k$ satisfies

$$\gamma^k_n + \rho \geqslant \sum_{e \in E} f_e(\overline{z}^k(e)) = \delta^k.$$

Since γ^k_n is a valid lower bound, the above inequality implies that $\overline{z}^k$ is a ρ-optimal flow.

It remains to prove that the linear approximation method is finite. Suppose the contrary, namely that we get an infinite sequence $\{z^k_0\}$. Since the solution x^k_ℓ to the approximate subproblem in stage ℓ corresponds to a vertex of the simplex defined by the constraints, the feasible flow $z^k_0 = (x^k_1, x^k_2, \ldots, x^k_n)$ corresponds to a spanning tree. Since the number of spanning trees is finite, there exist a feasible flow $z^* = (x^*_1, x^*_2, \ldots, x^*_n)$ and an infinite subsequence $\{z^{k_\theta}_0\}$ such that

$$z^{k_\theta}_0 = z^* \quad \text{for all } \theta.$$

This implies

$$s_\ell^{k_\theta} \;=\; s_\ell^* \qquad \text{for all } \theta \text{ and all } \ell = 1, 2, \dots, n-1. \tag{12}$$

Obviously, for $\ell = 0$ one has

$$H_0(s_0^*) \;=\; H_0^{k_\theta}(s_0^*) \qquad \text{for all } \theta \geqslant 1.$$

Now suppose that for some $\ell \in \{0, 1, \dots, n-1\}$ one has

$$H_\ell(s_\ell^*) \;=\; H_\ell^{k_\theta}(s_\ell^*) \qquad \text{for all } \theta \geqslant \ell + 1. \tag{13}$$

We are going to prove that

$$H_{\ell+1}(s_{\ell+1}^*) \;=\; H_{\ell+1}^{k_\theta}(s_{\ell+1}^*) \qquad \text{for all } \theta \geqslant \ell + 2. \tag{14}$$

Indeed,

$$\begin{aligned} H_{\ell+1}(s_{\ell+1}^*) &= H_{\ell+1}(s_{\ell+1}^{k_{\ell+1}}) \qquad \text{(by (12))} \\ &= \min \left\{ H_\ell(s_\ell) + \sum_{e \in E_t(v_{\ell+1})} f_e(x_{\ell+1}(e)) \;\middle|\; \sum_{e \in E_t(v_{\ell+1})} x_{\ell+1}(e) = q_{s_{\ell+1}^{k_{\ell+1}}}(v_{\ell+1}),\; x_{\ell+1} \geqslant 0 \right\}, \end{aligned}$$

where s_ℓ is defined as in the approximation subproblem in stage $\ell+1$ and iteration $k_{\ell+1}$. So, for $\theta \geqslant \ell + 2$ one has

$$\begin{aligned} H_{\ell+1}(s_{\ell+1}^*) &\leqslant H_\ell(s_\ell^{k_{\ell+1}}) + \sum_{e \in E_t(v_{\ell+1})} f_e(x_{\ell+1}^{k_{\ell+1}}) \\ &= \gamma_{\ell+1}^{k_{\ell+1}} \\ &\leqslant H_{\ell+1}^{k_{\ell+1}+1}(s_{\ell+1}^{k_{\ell+1}}) \qquad (\text{since } (\gamma_{\ell+1}^{k_{\ell+1}}, s_{\ell+1}^{k_{\ell+1}}) \in \text{hypo} H_{\ell+1}^{k_{\ell+1}+1}) \\ &\leqslant H_{\ell+1}^{k_\theta}(s_{\ell+1}^{k_{\ell+1}}) \qquad (\text{since } k_\theta \geqslant k_{\ell+1} + 1) \\ &= H_{\ell+1}^{k_\theta}(s_{\ell+1}^*) \qquad \text{(by (12))} \\ &\leqslant H_{\ell+1}(s_{\ell+1}^*). \qquad \text{(by Lemma 4.4)} \end{aligned}$$

Thus at iteration k_n we have

$$H_\ell(s_\ell^{k_n}) \;=\; H_\ell^{k_n}(s_\ell^{k_n}) \qquad \text{for all } \ell = 1, 2, \dots, n-1. \tag{15}$$

Therefore,

$$\begin{aligned}\gamma_n^{k_n} &= H_{n-1}^{k_n}(s_{n-1}^{k_n}) + \sum_{e\in E_t(v_n)} f_e(x_n^{k_n}(e))\\ &= H_{n-1}(s_{n-1}^{k_n}) + \sum_{e\in E_t(v_n)} f_e(x_n^{k_n}(e)) \qquad \text{(by (15))}\\ &\geqslant \min\left\{ H_{n-1}(s_{n-1}) + \sum_{e\in E_t(v_n)} f_e(x_n(e)) \right|\\ &\qquad\qquad \left. \sum_{e\in E_t(v_n)} x_n(e) = q(v_n),\ x_n \geqslant 0 \right\},\end{aligned}$$

where $s_{n-1}(v) = x_n(v, v_n)$, $v \in C_{n-1}$. The last minimum is nothing but the optimal value of our problem. So, $\gamma_n^{k_n}$ is the optimal value. It can be seen from (15) that

$$\gamma_n^{k_n} = \sum_{e\in E} f_e(z_0^{k_n}(e)),$$

i.e., $\gamma_n^{k_n}$ must coincide with the upper bound δ^{k_n}. Thus the linear approximation method terminates at iteration k_n. This is a contradiction to the assumption that sequence $\{z_0^{k_\theta}\}$ is infinite. This completes the proof of the convergence theorem. □

5. Computational results

As mentioned in the beginning, a dynamic programming approach for the minimum concave cost network flow problem seems only to be reasonable, if the vertices in graph G have small degrees. Thus we tested the linear approximation method mainly on several such graph classes, the largest instance being a graph with 1103 vertices and 2202 arcs.

The only recent computational results for large scale uncapacitated minimum concave cost network flow problems in the literature of which we are aware, are the results given by Holmqvist, Migdalas and Pardalos [7] and Kim and Pardalos [8]. In [7] instances up to 500 vertices and 7500 arcs are solved by using a GRASP type heuristics. The authors do not report on the quality of their suboptimal solutions. In [8] large scale fixed charge network flow problems are treated up to 202 vertices and and 10200 arcs. However, only problems up to 335 arcs are solved optimally.

In all tests below, the demands $q(v)$ are uniformly distributed random numbers drawn from the interval [0.1, 10000]. For the cost function f_e on an arc e we always assume the form:

$$f_e(x(e)) = \begin{cases} 0 & \text{if } x(e) = 0\\ \alpha_1 + \alpha_2 \cdot x(e) + \alpha_3 \cdot x(e)^{\alpha_4} & \text{if } x(e) > 0,\end{cases}$$

where α_1, α_2, α_3 are uniformly distributed random numbers in $[0, 100]$ and α_4 is a uniformly distributed random number in $[0, 1]$. The approximation method is implemented in ANSI C and the codes are compiled using the *cc* compiler. The linear programs are solved by using the callable library from CPLEX 6.0.2. All tests are run on a HP J200 PARISK 7200 workstation with 1.5 GB of RAM.

For every vertex v_l we create a basic data set for the n_l linear programs corresponding to the arcs with the terminal vertex v_l. In an iteration we justify the right-hand side in the data set to obtain a linear program to be solved.

In order to solve the linear programs we have always used the CPLEX primal simplex algorithm. By performing experimental tests we have seen that applying the CPLEX primal algorithm is better than applying the CPLEX dual algorithm to the modified problems. This can be explained by the fact that CPLEX is performing internally a lot of preprocessing and reduction steps which considerably reduce the problem size and lead to fast reoptimization.

In all our computational experiments we use $\rho = 10^{-6}$ as a stopping criterion. That is we virtually ask for the true optimal solution. Moreover, we do not use any local search in Step 3 of the algorithm for improving the upper bound. Since the gap between lower and upper bounds of the optimal flow value is quite large during the starting iterations, and we virtually ask for the optimal solution, local search could be time consuming to use. From computational experiments we see that usually the upper bound reaches the optimal value before the lower bound. This happens just very few iterations before we stop the algorithm. Hence, in this case a better upper bound during the iterations obtained by local search is not helpful. But we think local search may be helpful when an approximate solution is asked for. In this case one may start using local search after a given number of iterations (i.e., when the obtained solution is near enough to the optimal solution).

The number of linear programs solved in each iteration is equal to the number of arcs in graph G. The total number of linear programs solved in our computational experiments includes the linear programs solved in order to discard redundant variables by using condition (4).

Since all test runs took a relatively large number of iterations (usually more than 1000), we discard redundant variables by using condition (4) after every 500 iterations. Although we have to solve additional linear programs in order to test condition (4), this pays off in terms of computational time and memory used. We have seen from experimental tests that good results can be obtained by performing the test of condition (4) every 500 iterations.

The first graph class we consider is specially layered graphs of the following form. The graphs have p layers and in every layer r vertices. The source is connected with all vertices in the first layer. For $r = 2$ we connect always all vertices of two adjacent layers, in the cases $r \geqslant 3$ we generate the arcs randomly. Figure 1 shows the structure of such a layered graph and Table 1 reports computational results for this graph class.

In the table below are given the computational results for the layered graphs.

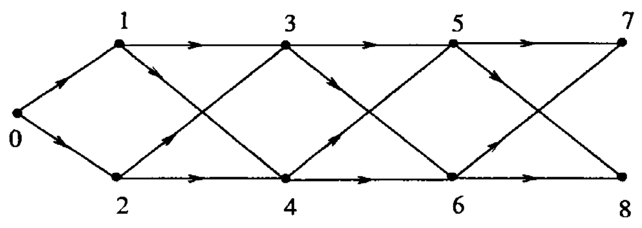

Figure 1. A layered graph ($r = 2$, $p = 4$).

Table 1. Results for the layered graphs

No. of vertices per layer	No. of vertices	No. of destinations	Average No. of arcs	No. of tests	Average No. of iterations	Average CPU (s)
2	23	22	42.0	5	75.0	2.58
2	53	52	102.0	5	181.4	21.70
2	103	102	202.0	5	357.2	117.31
2	203	202	402.0	5	728.4	807.60
2	303	302	602.0	2	1062.5	3276.73
2	403	402	802.0	2	1417.0	4581.85
2	503	502	1002.0	1	1795.0	9503.58
2	603	602	1202.0	1	2137.0	13460.33
2	803	802	1602.0	1	2849.0	27935.86
2	1003	1002	2002.0	1	3643.0	58424.93
2	1103	1102	2202.0	1	3923.0	67854.50
3	34	33	77.2	5	378.4	62.74
3	64	63	157.2	5	791.8	452.15
3	154	153	379.0	2	2016.0	3989.01
3	304	303	743.0	1	4063.0	20910.34
4	45	44	123.0	2	1919.0	1588.78
4	105	104	300.0	1	5511.0	15193.96

We perform a number of tests for a given structure of the graph for a given number of vertices. The average number of arcs, average number of iterations and the average CPU times (in seconds) are given in the table. Recall that for graphs with 2 vertices per layer the number of arcs is the same for each test of a given instance, while for graphs with more than two vertices per layer the number of arcs may be different in each of the tests. The CPU time for each problem in the table represents the computational time to solve the problem, without taking into consideration the generation of data.

Second, we test the algorithm on acyclic graphs as shown in Figure 2. We generate these graphs due to the condition that the head of an arc is at most 3 layers away from the layer which contains the tail of this arc and that the indegree

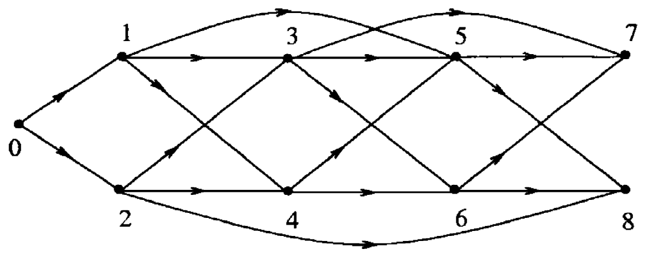

Figure 2. An acyclic graph of the second test-class.

Table 2. Results for the acyclic graphs

No. of vertices per layer	No. of vertices	No. of destinations	Average No. of arcs	No. of tests	Average No. of iterations	Average CPU (s)
2	23	22	50.4	5	143.0	9.20
2	33	32	76.4	5	351.2	55.70
2	53	52	124.4	5	447.6	123.00
2	103	102	248.2	5	1124.6	1224.79
2	203	202	492.5	2	2319.5	5677.41
2	303	302	748.0	1	3177.0	15553.04
3	16	15	36.2	5	315.4	27.10
3	34	33	90.5	2	1649.0	1107.08
3	64	63	175.0	1	4250.0	7331.95

of the vertices is not too high. Table 2 shows computational results for this class of graphs.

As a third graph class we consider transportation networks with one source, m transshipment vertices and r sinks. The source is connected with every transshipment vertex and these are connected with all sinks (cf. Figure 3). Thus these graphs have $m(r+1)$ arcs in total. Table 3 shows the computational results for this graph class.

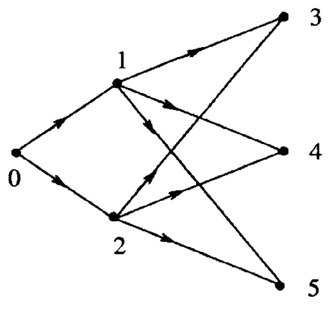

Figure 3. A transportation network with 2 transshipment vertices and 3 sinks.

Table 3. Results for the transportation graphs

No. of vertices	No. of transship. vertices	No. of destinations	No. of arcs	No. of tests	Average No. of iterations	Average CPU (s)
53	2	52	102	5	441.6	81.63
103	2	102	202	5	1111.8	903.62
203	2	202	402	1	3281.0	7796.14
34	3	33	93	5	713.0	206.99
54	3	53	153	5	1717.0	1085.41
104	3	103	303	1	7182.0	19668.83
15	4	14	44	5	256.0	16.52
25	4	24	84	5	957.0	294.75
55	4	54	204	1	6460.0	13922.85
16	5	15	55	5	563.6	110.58
21	5	20	80	5	1205.0	431.60
26	5	25	105	2	2042.5	1167.84

If we compare the results for graphs with different structures but with approximately the same number of vertices and arcs it can be seen that the algorithm performs better for the layered network. For example, if the first 3 instances of transportation graphs in the above table are compared with the corresponding layered graphs with 2 vertices per layer (see Table 1), it can be seen that the number of iterations and the CPU times are smaller in the case of a layered network. From the tables it is clear that the results are better for graphs with small degree vertices and small cardinality $|C_l|$.

Finally, we consider complete, acyclic graphs with up to 21 vertices, (see Figure 4). The corresponding computational results can be found in Table 4.

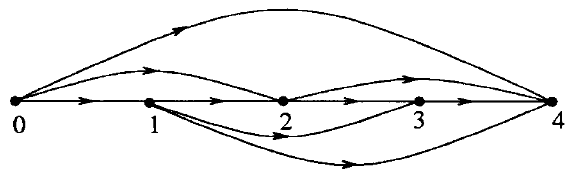

Figure 4. A complete graph.

It can be observed that the above method is not practical for complete graphs. For graphs with 19 or more vertices the number of iterations and computation time is exorbitant.

Table 4. Results for complete graphs

No. of Vertices	No. of Destinations	No. of Arcs	No. of Tests	Average No. of Iterations	Average CPU (s)
11	10	55	5	187.6	13.20
13	12	78	5	427.6	85.03
16	15	120	5	1132.6	725.35
19	18	171	1	5149.0	17491.17
20	19	190	1	6393.0	26220.21
21	20	210	1	7822.0	47994.34

6. Conclusions

The computational results show that dynamic programming might successfully be applied to minimum concave cost network flow problems for layered networks whose vertices have a small indegree. The proposed successive linear underestimation method for the involved concave cost functions has the nice feature that several times linear programs with the same coefficient matrix have to be solved. Two linear programs in two successive iterations differ only in one column of the coefficient matrix and in the right hand side. Thus one can take advantage of these properties by applying appropriate techniques from linear programming.

The advantage of this method is that the quality of the solution can be controlled. We have virtually found the true optimal solution in all our computational experiments.

Instead of linear underestimations of the involved functions it might be worthwhile to consider piecewise linear approximations for the convex cost functions like in the paper of Kim and Pardalos [8]. Such an approach is left to a future study.

Acknowledgements

We thank Bettina Klinz and Eranda Çela for many useful discussions on this topic. Moreover, we thank two anonymous referees for their careful attention.

References

1. Erickson, R.E., Monma, C.L. and Veinott Jr., A.F. (1987), Send-and-split method for minimum-concave-cost network flows, *Mathematics of Operations Research* 12: 634–664.
2. Falk, J.E. and Hoffman, K.L. (1976), A successive underestimation method for concave minimization problems, *Mathematics of Operations Research* 1: 251–259.
3. Ford Jr., L.R. and Fulkerson, D.R. (1962), *Flows in Networks*, Princeton University Press, Princeton, New Jersey.

4. Gallo, G., Sandi, C. and Sodini, C. (1980), An algorithm for the min concave cost flow problem, *European Journal of Operational Research* 4: 248–259.
5. Guisewite, G.M. and Pardalos, P.M. (1990), Minimum concave-cost network flow problems: applications, complexity, and algorithms, *Annals of Operations Research* 25: 75–100.
6. Guisewite, G.M. and Pardalos, P.M. (1991), Algorithms for the single-source uncapacitated minimum concave-cost network flow problem. *Journal of Global Optimization* 1: 245–265.
7. Holmqvist, K., Migdalas, A. and Pardalos, P.M. (1998), A GRASP algorithm for the single source uncapacitated minimum concave-cost network flow problem, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 40: 131–142.
8. Kim, D. and Pardalos, P.M. (1999), A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure, *Operations Research Letters* 24: 195–203.
9. Kim, D. and Pardalos, P.M. (2000), Dynamic slope scaling and trust interval techniques for solving concave piecewise linear network flow problems, *Networks* 35: 216–222.
10. Klinz, B. and Tuy, H. (1993), Minimum concave-cost network flow problems with a single nonlinear arc cost, in: P. Pardalos and D. Du (eds.), *Network Optimization Problems*, World Scientific, pp. 125–143.
11. Tuy, H. (2000), The minimum concave-cost network flow problem with a fixed number of nonlinear arc costs: complexity and approximation, in: P.M. Pardalos (ed.), *Approximation and Complexity in Numerical Optimization: Continuous and Discrete Problems*, Kluwer Academic Publishers, pp. 383–402.
12. Tuy, H., Ghannadan, S., Migdalas, A. and Värbrand, P. (1996), A strongly polynomial algorithm for a concave production-transportation problem with a fixed number of nonlinear variables, *Mathematical Programming* 72: 229–258.
13. Wagner, H.M. (1975), *Principles of Operations Research*, 2nd edition, Prentice–Hall, Inc., Englewood Cliffs, New Jersey.
14. Zangwill, W.I. (1968), Minimum concave cost flows in certain networks, *Management Science* 14: 429–450.