ZOR – Methods and Models of Operations Research (1993) 37:31–58

The Quickest Flow Problem

RAINER E. BURKARD, KARIN DLASKA, AND BETTINA KLINZ

Institut für Mathematik, TU Graz, Kopernikusgasse 24, 8010 Graz, Austria

Abstract: Consider a network $\mathcal{N} = (G, c, \tau)$ where $G = (N, A)$ is a directed graph and c_{ij} and τ_{ij}, respectively, denote the capacity and the transmission time of arc $(i, j) \in A$. The quickest flow problem is then to determine for a given value v the minimum number $T(v)$ of time units that are necessary to transmit (send) v units of flow in $\mathcal{N}$ from a given source s to a given sink s'.

In this paper we show that the quickest flow problem is closely related to the maximum dynamic flow problem and to linear fractional programming problems. Based on these relationships we develop several polynomial algorithms and a strongly polynomial algorithm for the quickest flow problem.

Finally we report computational results on the practical behaviour of our metholds. It turns out that some of them are practically very efficient and well-suited for solving large problem instances.

1 Introduction

Flow Problems on networks belong to the most studied problems of mathematical programming. They have numerous applications in practice since highway, rail, electrical, communication and many other physical networks pervade our everyday lives. In many applications of flow problems not only *the amount of flow* to be transmitted but also the *time* needed for the transmission plays an essential role. This leads to so-called *dynamic flow problems* where a *transmission* (*traversal*) time is attached to each link (arc) of the network. (See also the classical work of Ford and Fulkerson [15], and the survey paper of Aronson [5].)

In our paper we treat a special type of dynamic flow problems. Let $G = (N, A)$ be a directed graph with node set N and arc set A and $\mathcal{N} = (G, c, \tau)$ be the associated network where a capacity c_{ij} and a nonnegative integer transmission time τ_{ij} are attached to every arc $(i, j) \in A$. Thereby c_{ij} represents the maximum number of flow units that can be sent on arc (i, j) per time unit, and if f_{ij} units of

Partial financial support by the Air Force Office of Scientific Research under grants AFOSR-89-0512 and AFOSR-90-0008 is gratefully acknowledged by the first author.

0340–9422/93/1/31–58 $2.50

flow start at time $t \in \mathbb{N}_0$ in node i and are sent to node j along the arc (i, j), they arrive there at time $t + \tau_{ij}$. Adopting the usual convention, we denote the number of nodes in N by n and the number of arcs in A by m. Further, let the source s and the sink s' be two distinguished nodes in N.

The *quickest flow problem (QFP)* now consists in finding in $\mathcal{N}$ a (dynamic) flow f of given value v from s to s' which sends the given v units of flow from s to s' in the minimum number $T(v)$ of time units. The flow f has to obey the flow conservation constraints in all intermediate nodes ($\neq s, s'$) and the capacity constraints on all arcs. Both groups of constraints have to hold for the whole transmission period, i.e. for $t = 0, 1, \ldots, T(v)$.

Possible areas of application of this model include evacuation problems (see e.g. Hamacher [19] and Chalmet, Francis and Saunders [11]) and data transmission problems in communication networks.

A similar problem, the *quickest path problem (QPP)*, was recently treated by Chen and Chin [12] and Rosen, Sun and Xue [36]. These authors were interested in a *path* in $\mathcal{N}$ from s to s' along which the given amount v of flow can be transmitted in the shortest possible time. Whereas in the *(QPP)* only *one path* can be used for the transmission of flow, we allow in the *(QFP)* *simultaneous transmission* of flow on *different* but not necessarily (node- and arc-) disjoint paths from the source to the sink. The latter problem is addressed in Burkard, Dlaska and Kellerer [8].

This paper is organized as follows: In Section 2 we set up a mathematical model for *(QFP)* and relate it to the classical maximum dynamic flow problem. Motivated by the close relationship between maximum dynamic and quickest flows we study in Section 3 the value of a maximum dynamic flow for the given time interval $[0, T]$ as a function of T. In Section 4 we first describe two pseudopolynomial-time algorithms for finding quickest flows in networks, and then we propose several polynomial-time algorithms which are based on the results of Section 3. The computational results presented in Section 5 show that some of these algorithms seem also to be well suited for solving the *(QFP)* in practice. Theoretically, one can do even better. So, we derive in Section 6 a strongly polynomial-time algorithm based on results of Megiddo [28, 29]. Finally, we close the paper with some concluding remarks in Section 7.

2 Quickest Flows

Let $G = (N, A)$ be a given directed graph and let $\mathcal{N} = (G, c, \tau)$ be the associated network with source s and sink s', integer capacities c_{ij} and nonnegative integer transmission times τ_{ij}. Further, let $I_T := [0, T]$ be a discrete time interval (containing only integer-valued points) with $T \in \mathbb{N}_0$ and let denote N_s the set $N \setminus \{s, s'\}$.

Definition 1: A *dynamic flow* of value v, $v \in \mathbb{N}$, from s to s' is a mapping $f: A \times I_T \to \mathbb{N}_0$ fulfilling the following properties

$$\sum_{t=0}^{T} \left(\sum_{(s,j) \in A} f_{sj}(t) - \sum_{(i,s) \in A} f_{is}(t - \tau_{is}) \right) = v \tag{1}$$

$$\left(\sum_{(i,j) \in A} f_{ij}(t) - \sum_{(k,i) \in A} f_{ki}(t - \tau_{ki}) \right) = 0 \qquad \forall i \in N_s \ , \quad \forall t \in I_T \tag{2}$$

$$\sum_{t=0}^{T} \left(\sum_{(s',j) \in A} f_{s'j}(t) - \sum_{(i,s') \in A} f_{is'}(t - \tau_{is'}) \right) = -v \tag{3}$$

$$0 \leq f_{ij}(t) \leq c_{ij} \qquad \forall (i,j) \in A \ , \quad \forall t \in I_T \tag{4}$$

where for notational convenience we assume throughout that $f_{ij}(t) = 0$ for $t < 0$. $f_{ij}(t)$ denotes the flow that leaves node i along arc (i, j) at time t. □

Equation (1) describes that the flow which leaves the source s in the time interval I_T totals v. Exactly this flow arrives during this time in the sink s', confer (3). The equations (2) describe that at any time t the flow which arrives in an intermediate node $i \neq s, s'$ also leaves this node at this time. (4) expresses the capacity constraints at any time t. (Note that our model is a discrete dynamic flow problem where a flow of $f_{ij}(t)$ is allowed to start at node i at any time $t \in I_T$.)

Definition 2: A *quickest flow* in $\mathcal{N}$ from s to s' for a given flow value v is a dynamic flow fulfilling (1)–(4) where $T = T(v)$ is minimum. Thus the quickest flow problem can be stated as

$$(QFP) \qquad \min T = T(v)$$

$$\text{s.t. } (1)\text{–}(4) \ . \qquad \square$$

The quickest flow problem is closely related to the *maximum dynamic flow problem (MDFP)* as introduced by Ford and Fulkerson [15]. The maximum dynamic flow problem asks for a dynamic flow of maximum value $v = v(T)$ in a given time interval $[0, T]$. In contrast to the *(QFP)*, this problem can be written as linear program. In a certain sense *(QFP)* is the inverse problem of *(MDFP)*. Hence it seems to make sense to investigate possible connections between quickest flows and maximum dynamic flows.

In the following we denote by $T(v)$ the transmission time of a quickest flow of given value v, $v \in \mathbb{N}$, and by $v(T)$ the value of a maximum dynamic flow in the given time interval $[0, T]$, $T \in \mathbb{N}_0$. For notational convenience we further define $v(-1) := 0$. The following straightforward lemma expresses the close relationship between maximum dynamic and quickest flows.

Lemma 1: Let f be a dynamic flow of value v in the time interval $[0, T]$, $T \geq 0$. If

$$v(T-1) < v \tag{5}$$

then f is a quickest flow of value v, and for the minimum transmission time $T(v)$ we get

$$T(v) = T\,. \tag{6}$$

Our approach for solving the quickest flow problem will mainly rely on Lemma 1. The minimum transmission time $T(v)$ can be determined as follows. From the definition of $v(T)$ it follows that $v \leq v(T(v))$. In view of the conditions (5) and (6) of Lemma 1 also $v(T(v)-1) < v$ must hold. Thus, $T(v)$ equals the minimum T such that $v(T) \geq v$. In order to compute the quickest flow itself, we determine a maximum dynamic flow $f^{T(v)}$ for the interval $[0, T(v)]$ and decrease its value to v in case that $v(T(v)) > v$.

In the theory of dynamic flows so-called *temporally repeated flows* are of special interest. Let $g\colon A \to \mathbb{N}_0$ be a static (classical) flow of value $|g|$ from s to s' in the network $\mathcal{N}$. (I.e. we require flow conservation at each intermediate node $\neq s, s'$ and the usual capacity constraints $g_{ij} \leq c_{ij}$ for all $(i, j) \in A$.) Now we decompose the flow g into flows $g_1, \ldots, g_r$ along paths $P_1, \ldots, P_r$ from the source s to the sink s'. Let $|g_k|$ designate the value of the flow along path P_k and $\tau(P_k)$ the transmission time of path P_k, i.e. the sum of the transmission times of the arcs along P_k. It is easy to see that by repeating the path flows g_k along the corresponding paths $(T - \tau(P_k) + 1)$ times, we get a feasible dynamic flow provided that $T \geq \max_{1 \leq k \leq r} \tau(P_k)$. A dynamic flow f obtained in this way is called a *temporally repeated flow* (TRF). A simple calculation yields that the value of this TRF f induced by the static flow g is equal to

$$v_f = \sum_{k=1}^{r} (T + 1 - \tau(P_k)) \cdot |g_k| = (T+1) \cdot |g| - \sum_{(i,j) \in A} \tau_{ij} \cdot g_{ij}\,. \tag{7}$$

According to Ford and Fulkerson [15] there is always a maximum dynamic flow that is temporally repeated. Based on formula (7), these authors have further shown that a maximum dynamic flow for a given period $[0, T]$ can be computed

efficiently by solving a minimum cost circulation problem. (Introduce a return arc (s', s) with capacity $c_{s's} = \infty$ and cost $d_{s's} = -(T+1)$ and define the remaining costs as $d_{ij} := \tau_{ij}$ for $(i, j) \in A$.) Denote this minimum cost circulation problem by $(MCCP)_T$.

The close relationship between maximum dynamic and quickest flows as described in Lemma 1, motivates now to take a closer look at the structure of the value $v(T)$ of a maximum dynamic flow, regarded as function of T, $T \in \mathbb{N}_0$.

3 The Value Function of Maximum Dynamic Flows

In this section we will frequently refer to static flows that induce a TRF of maximum value for a given time interval. Therefore, let $\mathcal{G}^T$ denote the class of all static flows that induce for the interval $[0, T]$ a TRF of maximum value $(= v(T))$. (In general we will have $|\mathcal{G}^T| > 1$ and the value of flows in the class $\mathcal{G}^T$ will not necessarily be unique.) Furthermore, let T_0 be the length of the shortest path from s to s' with respect to the transmission times. Obviously $T_0 = \min\{T \geq 0 : v(T) > 0\}$.

The following lemma is an immediate consequence of the foregoing discussion on temporally repeated flows.

Lemma 2: For all $T \geq T_0$ we have

$$v(T-1) < v(T) \ . \tag{8}$$

Proof: For $T = T_0$ relation (8) is an immediate consequence of the definition of T_0. Let $T > T_0$. Consider a static flow $g^{T-1} \in \mathcal{G}^{T-1}$ and denote again by $|g^{T-1}|$ its value. Obviously, g^{T-1} induces a TRF of value $v(T-1) + |g^{T-1}|$ for the interval $[0, T]$ since each path in the decomposition of g^{T-1} can be used once again (cf. (7)). As $v(T)$ is the value of a *maximum* dynamic flow for the interval $[0, T]$, we get $v(T) \geq v(T-1) + |g^{T-1}| > v(T-1)$. □

The following connection between $v(T+1)$, $v(T)$ and the values $|g^T|$ can now be obtained easily.

Lemma 3: Let $g^T \in \mathcal{G}^T$ and $g^{T+1} \in \mathcal{G}^{T+1}$. Then the following two properties hold for all $T \geq 0$:

$$|g^T| \leq |g^{T+1}| \tag{9}$$

and

$$v(T) + |g^T| \leq v(T+1) \leq v(T) + |g^{T+1}| \ . \tag{10}$$

Observe that (9) holds independently of the choice of the flows g^T and g^{T+1}. (Recall that the value of flows in the class $\mathcal{G}^T$ is not necessarily unique.)

Proof: The monotonicity of $|g^T|$ is a direct consequence of formula (7). The left part of relation (10) has already been shown above.

To prove the right hand side of (10), we apply the same kind of argument as in the proof of Lemma 2, but now in the other direction. Consider a static flow $g^{T+1} \in \mathcal{G}^{T+1}$. g^{T+1} induces a TRF of value $v(T+1) - |g^{T+1}|$ for the interval $[0, T]$ (confer again formula (7)). Now we obviously must have

$$v(T+1) - |g^{T+1}| \leq v(T) \ .$$

Rearranging yields the relation to be proven. □

From Lemma 3 it follows that some kind of convexity property holds for $v(T)$. For $T > 0$ let $\Delta(T)$ denote the difference $v(T) - v(T-1)$. Further let g_{max} be an arbitrary maximum (static) flow in the network $\mathcal{N}$ and denote by $|g_{max}|$ its value. By combining Lemma 2 and Lemma 3, we can now show:

Theorem 1:

(i) *The value function $v(T)$ of a maximum dynamic flow is a monotone increasing function. For $T \geq T_0$ it even increases strictly.*

(ii) *$\Delta(T)$ is monotone increasing, i.e. for all $T > 0$ we have*

$$\Delta(T+1) \geq \Delta(T) \ . \tag{11}$$

(iii) *$\Delta(T)$ can attain only values from the set $\{0, 1, \ldots, |g_{max}|\}$.*

Proof: Property (i) has already been proven in Lemma 2. To show (ii), note that relation (10) yields the following bounds for $\Delta(T)$, $T > 0$, where $g^T \in \mathcal{G}^T$ and $g^{T-1} \in \mathcal{G}^{T-1}$:

$$|g^{T-1}| \leq \Delta(T) \leq |g^T| \ . \tag{12}$$

Observing relation (9) completes now the proof of (ii).

Since g^T is a static flow on $\mathcal{N}$ we obviously have $0 \leq |g^T| \leq |g_{max}|$ for all $T \geq 0$, so (12) implies $0 \leq \Delta(T) \leq |g_{max}|$. Thus property (iii) follows from the integrality of the values $\Delta(T)$. □

Remark 1: Regarding T as a parameter in the minimum cost circulation problem $(MCCP)_T$ introduced in Section 2 yields a special parametric cost linear programming problem. $v(T)$ equals then the negative of the optimal value function of problem $(MCCP)_T$ and the slope of the optimal value function equals the negative of the flow on the return arc (cf. formula (7)). Since the optimum value function of such a parametric problem is continuous, piecewise linear and concave (see e.g. Murty [31]), all properties of $v(T)$ mentioned in Theorem 1 can now be obtained in a non-combinatorial way as well. Of course, we are still interested only in the values of $v(T)$ for integer arguments, but treating this function as a continuous instead of as a discrete function simplifies many of the definitions and arguments of the next section, and therefore we will adopt to this convention for the rest of the paper.

The following result will be helpful for solving *(QFP)*:

Lemma 4:

(i) *Let* $g^{\hat{T}} \in \mathcal{G}^{\hat{T}}$ *for* $\hat{T} \geq 0$. *Then* $|g^{\hat{T}}|$ *is a subgradient of* $v(T)$ *at* $T = \hat{T}$.

(ii) *Further, if* $\bar{g}^{\hat{T}} \in \arg\min\{|g|: g \in \mathcal{G}^{\hat{T}}\}$ *and* $\hat{T} > 0$ *then*

$$v(\hat{T}) = v(\hat{T} - 1) + |\bar{g}^{\hat{T}}| \ . \tag{13}$$

This means that $|\bar{g}^{\hat{T}}|$ *is the left-hand-side derivative of* $v(T)$ *at* $T = \hat{T}$, *or in other words* $\Delta(\hat{T}) = |\bar{g}^{\hat{T}}|$.

Proof: From formula (7) we know that $v(\hat{T}) = (\hat{T} + 1) \cdot |g| - \sum_{(i,j) \in A} \tau_{ij} \cdot g_{ij}$ for any static flow $g \in \mathcal{G}^{\hat{T}}$. It follows directly from the notion of a subgradient that $|g^{\hat{T}}|$ is indeed a subgradient of $v(T)$ at $T = \hat{T}$. Assertion (ii) is an immediate consequence of the fact that the left-hand-side derivative at $T = \hat{T}$ is simply the smallest subgradient at this point. □

Up to now we still do not know how to compute a quickest flow. But from Lemma 1 and the subsequent discussion we know that in order to solve the quickest flow problem for a given flow value v, we have to find the *minimum integer* T for which $v(T) \geq v$ holds. This can now be done efficiently by using the properties of $v(T)$ stated in Theorem 1. The details will be given in the next section.

4 Algorithms for Finding Quickest Flows

The key problem underlying the computation of the minimum transmission time is a typical parametric search problem: We have given a problem in the formulation of which a parameter occurs and try to find the smallest (largest) value of this parameter such that a certain property is fulfilled. Problems of this type have already been dealt with many times in literature. One field of application that is particularly important from the point of view of this paper is the rather popular parametric approach in fractional programming (see e.g. Ibaraki [21] and Megiddo [28]). This close relationship between fractional programming and the *(QFP)* is no mere coincidence since *(QFP)* can indeed be formulated as a linear fractional program. (T can be written as quotient of two linear functions of the flow variables g_{ij}, cf. (7).)

So some of the ideas we are going to propose can already be found in the literature on the parametric approach in fractional programming. (For details the interested reader is referred to the papers by Ibaraki [21] and Schaible and Ibaraki [40], the monograph by Schaible [39] and the references given therein.) What we try here is first to emphasize the combinatorial side of the *(QFP)* and secondly to use additional properties of the *(QFP)* which accelerate the general algorithms.

In the following we will first describe two pseudopolynomial algorithms for the *(QFP)*, where the second of these algorithms is already known from the literature. Thereafter we will propose several polynomial algorithms, and in Section 6 we shall even point out a strongly polynomial method.

4.1 Pseudopolynomial Algorithms

Since $v(T)$ is an increasing function, one approach for computing the minimum integer T for which $v(T) \geq v$ holds, would of course be to solve the parametric minimum cost circulation problem $(MCCP)_T$ starting from $T = 0$ and to stop as soon as $v(T) \geq v$. Unfortunately, by adding a return arc (s', s) with infinite capacity and cost $-(T + 1)$ to the famous pathological networks of Zadeh [42], we get for each $k \geq 2$ an instance of type $(MCCP)_T$ with $n = 2k + 2$ nodes, $b = 2^k - 3$ breakpoints (=slope changes of the optimal value function) and maximum static flow value $|g_{max}| = 2^k + 2^{k-2} - 2$. (Note that both b and $|g_{max}|$ are exponential in n.) These facts can be proved by similar ideas as in Carstensen [10] and Ruhe [38]. It follows that the parametric approach suggested above yields only a pseudopolynomial method for *(QFP)*.

Another pseudopolynomial algorithm for the *(QFP)* has already been described in Hamacher [19] and is based on the following results: A dynamic flow for the interval $[0, T]$ is termed *earliest arrival flow* (EAF) if the amount of flow

that has already arrived in the sink s' until time t is maximum for all t, $t = 0, 1, \ldots, T$. An EAF for the time interval $[0, T]$ is always a quickest flow for the value $v = v(T)$, see Jarvis and Ratliff [23]. (Note the similarities to Lemma 1.) We now start to compute an EAF by a method given by Minieka [28]. As soon as for the first time a dynamic flow with value $\geq v$ is arrived, we terminate and determine the minimum T such that the EAF for $[0, T]$ has value $\geq v$. This T obviously equals the minimum transmission time $T(v)$. As up to $|g_{max}|$ shortest augmenting paths have to be found, where again $|g_{max}|$ is the value of a maximum static flow, this approach yields again only a pseudopolynomial algorithm for the *(QFP)*.

In the sequel we will suggest several classes of polynomial-time algorithms for solving the quickest flow problem.

4.2 *Polynomial Search Methods on the T-Axis*

Since $v(T)$ is monotone increasing and T attains only integer values, a binary search method with respect to T suggests itself. One starts with an interval $[T_l, T_u]$ such that $v(T_l) < v$ and $v(T_u) > v$, thus $T(v) \in [T_l, T_u]$. Then one computes the mid point of this interval, say T_c, and tests by computing $v(T_c)$ whether one is already finished or has to continue the search to the left respective to the right of T_c.

Although by using this method the search interval gets halved at each iteration, the convergence of this method is rather slow in practice. In the sequel we will describe how the decrease of the interval can be accelerated by exploiting the convexity of $v(T)$.

4.2.1 An Improvement of Pure Binary Search

To simplify the exposition, we first introduce the following notations (see Fig. 1): Let L_v be the line parallel to the T-axis at height v, where v is the value of the quickest flow to be determined. Assume further that $T(v) \in [T_l, T_u]$. Then we denote by $\gamma(T_l, T_u)$ the value of T where the chord $ch(T_l, T_u)$ through the points $(T_l, v(T_l))$ and $(T_u, v(T_u))$ crosses the line L_v. Further, let us denote by $L_{supp}(\hat{T})$ an arbitrary supporting line to $v(T)$ at the point $(\hat{T}, v(\hat{T}))$ and by $\sigma(L_{supp}(\hat{T}))$ the T-coordinate of the intersection point of this supporting line with the line L_v. (If $v(T)$ is differentiable at point $(\hat{T}, v(\hat{T}))$, then the tangent at this point is the unique supporting line, otherwise any subgradient at this point can be taken as slope of the supporting line $L_{supp}(\hat{T})$.)

Assume it is known that $T(v) \in [T_l, T_u]$. Then the following improved bounds follow immediately from the convexity of $v(T)$ (see also Figure 1):

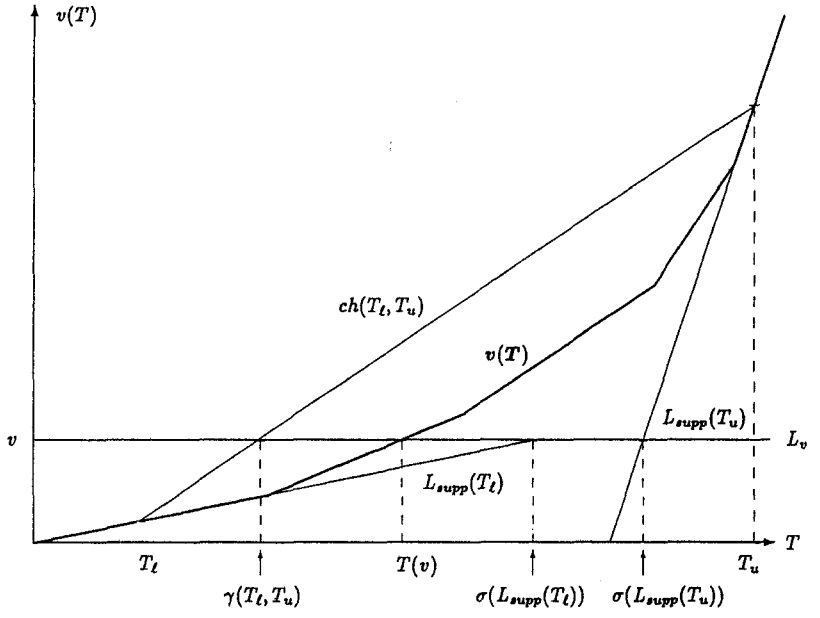

Fig. 1. Improved lower and upper bounds: $v(T)$ is drawn as bold line

$$\lceil \gamma(T_l, T_u) \rceil \leq T(v) \tag{14}$$

and

$$T(v) \leq \lceil \min\{\sigma(L_{supp}(T_l)), \sigma(L_{supp}(T_u))\} \rceil \ . \tag{15}$$

Observe that bound (15) is obtained from a Newton-like step and bound (14) is stimulated by the regula falsi. These two bound improvement steps enormously improve the practical efficiency of pure binary search, provided they are applied in each iteration of the binary search. In the following some aspects of the algorithm sketched above will be discussed in some more detail.

In (15) supporting lines to $v(T)$ at $T = T_l$, respective at $T = T_u$, and hence subgradients of $v(T)$ at these points are needed. Using the right-hand-side (the left-hand-side) derivative of $v(T)$ at $T = T_l$ (at $T = T_u$) obviously leads to the tightest upper bounds, but has the disadvantage of requiring a substantially larger computational effort than using the subgradient that results as a by-product in the evaluation of $v(T)$ (confer Lemma 4) for free.

As initial values for T_l and T_u respectively, we can take

$$T_l := \max\left\{T_0, \left\lceil \frac{v - v(0)}{|g_{max}|} \right\rceil\right\} \tag{16}$$

and

$$T_u := T_l + \left\lceil \frac{v - v(T_l)}{|g^l|} \right\rceil , \tag{17}$$

where g^l denotes a static flow that induces a TRF of maximum value for $T = T_l$. The validity of the lower bound (16) follows from the definition of T_0 (no flow can reach the sink for $T < T_0$) and from the fact that $|g_{max}|$ is an upper bound on the differences $\Delta(T) = v(T) - v(T-1)$ which implies that at least $\lceil (v - v(0))/|g_{max}| \rceil$ time units are necessary for achieving $v(T) \geq v$. The correctness of the upper bound (17) can be seen by a similar argument. Since $|g^l|$ is a subgradient of $v(T)$ at $T = T_l$, it follows that $\Delta(T_l + 1) \geq |g^l|$. Due to the monotonicity of $\Delta(T)$, we hence need at most $\lceil (v - v(T_l))/|g^l| \rceil$ additional time units in order to increase $v(T)$ from its current value $v(T_l)$ to v.

Analysis of the Improved Binary Search Algorithm

(i) The computation of an initial interval $[T_l, T_u]$ for $T(v)$ by means of (16) and (17) essentially amounts to solving one shortest path problem and one maximum flow problem. Then two minimum cost circulation problems for computing $v(T_l)$ and $v(T_u)$ have to be solved.

(ii) In each step of the binary search a minimum cost circulation problem has to be solved in order to evaluate $v(T_c)$ for the midpoint $T_c = \lfloor (T_l + T_u)/2 \rfloor$. All other operations including the bound improvement steps can be performed in constant time.

(iii) It is straightforward to observe that there are at most $\min\{\lceil \log v \rceil + 1, |g_{max}|\}$ iterations. (The length of the search interval is at least halved in each iteration and the length of the initial interval is $\leq v$ as can be seen from (16) and (17). Furthermore, $v(T)$ has at most $|g_{max}|$ linear pieces of non-zero slope.)

Hence, we get $O(\min(\log v, |g_{max}|) \cdot MIN(n, m))$ as overall complexity of the above algorithm, where $MIN(n, m)$ denotes the number of steps that are necessary to solve a minimum cost circulation problem on a network with n nodes and m arcs. The currently best known strongly polynomial algorithm for the minimum cost circulation problem due to Orlin [32] basically solves $O(m \log n)$ shortest path problems w.r.t. nonnegative weights and yields therefore $MIN(n, m) = O(m \log n(m + n \log n))$.

The improved binary search algorithm determines the minimum transmission time $T(v)$ and an associated optimal static flow $g^{T(v)} \in \mathcal{G}^{T(v)}$. The quickest flow itself can then be found in $O(nm)$ additional steps. (Decompose $g^{T(v)}$ into at most m path flows and construct the associated temporally repeated flow. In case that

this TRF has value $> v$, a new dynamic flow of value v can easily be obtained in $O(m)$ time – e.g. by appropriately modifying the given TRF in the last transmission period.) Thus the *(QFP)* can be solved in $O(\min(\log v, |g_{max}|) \cdot MIN(n, m))$ time, i.e. in polynomial time.

4.2.2 An Interpolation-Based Algorithm

In large practical problems it occurs very rarely that $T(v)$ locates around the midpoint T_c. As remedy Ibaraki [21] proposed an interpolation approach which can be adapted to the *(QFP)* as follows: We compute an interpolation function $\hat{v}(T)$ for $v(T)$ and choose the minimum integer T for which $\hat{v}(T) \geq v$ as next test point T_c. By this approach one hopes to find better estimates for the minimum transmission time, but the price one has to pay for the practical improvement is the theoretical shortcoming that now one cannot guarantee any longer the $(\lceil \log v \rceil + 1)$ upper bound on the number of iterations.

Obviously, $\hat{v}(T)$ should share as many properties with $v(T)$ as possible. Hence the interpolation function $\hat{v}(T)$ should be a convex, monotone increasing, continuous function for which furthermore the function values and subgradients at both endpoints of the current search interval should coincide with the corresponding values of the original function $v(T)$. Suppose it is known that $T(v) \in [T_l, T_u]$. Further, let g^l and g^u be static flows such that the associated flow values $|g^l|$ and $|g^u|$ are subgradients of $v(T)$ for $T = T_l$ and $T = T_u$, respectively. After some empirical experimentations Ibaraki came up with the following form for $\hat{v}(T)$:

$$\hat{v}(T) := \begin{cases} -|g^u| \cdot (T_u - T) + a(T_u - T)^b + v(T_u) & \text{if } |g^u| \neq \Phi(v) , \\ -|g^u| \cdot (T_u - T) + v(T_u) & \text{otherwise} , \end{cases} \tag{18}$$

where

$$\Phi(v) := \frac{v(T_u) - v(T_l)}{T_u - T_l} , \qquad b := \frac{|g^l| - |g^u|}{\Phi(v) - |g^u|} , \qquad a := -\frac{\Phi(v) - |g^u|}{(T_u - T_l)^{b-1}} .$$

4.2.3 A Newton-Type Algorithm

Another very popular approach for solving non-linear equations is *Newton's method* which in our case works as follows: We choose a $T^{(0)}$ as initial guess for $T(v)$ (e.g. $T^{(0)} := T_l$ given in (16)). Then we iteratively compute a sequence $T^{(k)}$ according to

$$T^{(k+1)} := \sigma(L_{supp}(T^{(k)})) \qquad \text{for } k \geq 0 \ . \tag{19}$$

That means $T^{(k+1)}$ is set to the value of T for which the supporting line $L_{supp}(T^{(k)})$ crosses the horizontal line L_v. We can stop this process as soon as in some iteration q either $v(T^{(q)}) = v$ or $|T^{(q-1)} - T^{(q)}| < 1$ holds. Upon termination the minimum transmission time $T(v)$ is known to be equal to $\lceil T^{(q)} \rceil$.

How many iterations does this method need in the worst case? Obviously, the number of different non-zero slopes of $v(T)$ and hence $|g_{max}|$ provides a trivial upper bound for the number of iterations. But, observe that this bound is not polynomial as $|g_{max}|$ can in the worst case become as large as $n \cdot c_{max}$, where c_{max} denotes the largest of the capacities c_{ij}. However, by using similar arguments as McCormick and Ervolina [27] and Radzik [35] we get the following polynomial bound:

Lemma 5: Newton's method as described above needs at most

$$O(\log v + \log |g_{max}|) \tag{20}$$

iterations.

Proof: For $k \geq 0$ let $g^{(k)} \in \mathcal{G}^{T(k)}$ be the static flow that results from the computation of $v(T)$ at $T = T^{(k)}$ and assume that $|g^{(k)}|$ is chosen as slope of the supporting line $L_{supp}(T^{(k)})$. From (19) we now get $T^{(k)} - T^{(k+1)} = (v(T^{(k)}) - v)/(|g^{(k)}|)$ for $k \geq 1$. Noting that $v(T) = (T+1) \cdot |g| - \sum_{(i,j) \in A} \tau_{ij} \cdot g_{ij}$ for $g \in \mathcal{G}^T$ (confer formula (7)) this implies

$$\frac{v(T^{(k+1)}) - v}{v(T^{(k)}) - v} \leq 1 - \frac{|g^{(k+1)}|}{|g^{(k)}|} \ . \tag{21}$$

It follows from (21) that depending on whether the rate $|g^{(k+1)}|/|g^{(k)}|$ is $\geq 1/2$ or $\leq 1/2$, either the sequence $(v(T^{(k)}) - v)$ or the sequence $|g^{(k)}|$ decreases by at least 1/2. It takes at most $\lceil \log |g_{max}| \rceil$ iterations to decrease $|g^{(k)}|$ from its initial value $|g^{(1)}| \leq |g_{max}|$ to 1. Further, at most $\lceil \log(|g_{max}| \cdot v) \rceil + 1$ iterations are necessary to decrease $v(T^{(k)}) - v$ from its initial value $(v(T^{(1)}) - v) \leq v \cdot |g^{(1)}| \leq v \cdot |g_{max}|$ (confer formula (7)) to a value less than 1. Summing up yields the desired bound. □

Remark 2: By a more careful choice of $T^{(0)}$ the bound (20) can be improved to $O(\log \Delta_{max} + \log v)$ with $\Delta_{max} = \min\{v, |g_{max}|\}$ being an upper bound for $\Delta(T(v))$,

the slope of the line segment on which the minimum transmission time $T(v)$ is located. We simply have to choose $T^{(0)}$ such that there is a supporting line of slope Δ_{max} at $T^{(0)}$. Then obviously we have $|g^{(1)}| \leq \Delta_{max}$ and the improved bound follows immediately from the proof of the lemma above.

Recently Rote [37] has given a refined analysis of the Newton approach for general linear fractional programming. Applying his result to our case yields the following upper bound for the number of iterations: $O\left(\log v / \log\left(1 + \frac{\log v}{\log \Delta_{max}}\right)\right)$. Hence, for $v = |g_{max}|^{\omega(1)}$, Newton's algorithm is asymptotically faster than the binary search algorithm on the T-axis discussed before. This follows also from the work of Radzik [35].

Further search methods which have not been mentioned here, such as a method based on the *regula falsi* can be found in Schaible and Ibaraki [40].

4.3 *Searching on the Set of Differences*

As mentioned before we know that $\Delta(T(v))$ is bounded above by $\Delta_{max} = \min\{|g_{max}|, v\}$. Consequently, applying binary search to the set of differences (slopes) results in only $O(\log \Delta_{max})$ iterations as opposed to $O(\log v)$ iterations that are needed by the binary search approach on the T-axis discussed above.

Observe that an initial interval $[\Delta_l, \Delta_u]$ for $\Delta(T(v))$ can be determined rather easily. Let $[T_l, T_u]$ be an initial interval for $T(v)$ and compute static flows $g^l \in \mathscr{G}^{T_l}$ and $g^u \in \mathscr{G}^{T_u}$, respectively. Then set $\Delta_l := |g^l|$ and $\Delta_u := \min\{|g^u|, v\}$.

The key problem that remains to be solved is now to determine for a given difference (slope) Δ_c the interval of T-values for which $v(T)$ has slope Δ_c. In the following we will describe an efficient method for that very purpose. For the ease of exposition we start with the following definitions:

Definition 3: Let $\hat{\Delta} > 0$ be a given difference. Then we denote by $\mathscr{I}_{\hat{\Delta}} := (\underline{T}, \overline{T}]$ the interval of all T such that $\Delta(T) = \hat{\Delta}$ for all integer $T \in \mathscr{I}_{\hat{\Delta}}$. □

Definition 4: Let g be a static flow on the network $\mathscr{N}$ and let $d_{ij} := \tau_{ij}$ denote the cost of arc (i, j). Then we denote by $\mathscr{N}^g = ((N, A^g), c^g, d^g)$ the residual network with respect to the flow g. There A^g denotes the set of residual edges, and c_{ij}^g and d_{ij}^g are the residual capacity and residual cost of arc $(i, j) \in A^g$, respectively. □

If the interval $\mathscr{I}_{\hat{\Delta}}$ is empty, then the difference $\hat{\Delta}$ never occurs, i.e. the function $v(T)$ has no segment of slope $\hat{\Delta}$. Otherwise, it follows that $v(T)$ has slope $\hat{\Delta}$ for $T \in [\underline{T}, \overline{T}]$. The following lemma can be used to compute the interval $\mathscr{I}_{\hat{\Delta}}$ efficiently.

Lemma 6: Let g be a static flow on the network $\mathcal{N}$ which has minimum cost among all flows of value $\hat{\Delta} > 0$, where the costs d_{ij} are given by the transmission times τ_{ij}. Further, let P_1 be a shortest path from the source s to the sink s' in $\mathcal{N}^g$ with respect to the costs d_{ij}^g and P_2 be a shortest path from s' to s, respectively. Then the bounds of the interval $\mathcal{I}_{\hat{\Delta}}$ can be computed as follows:

$$\underline{T} := -\sum_{(i,j)\in P_2} d_{ij}^g - 1 \qquad \text{and} \qquad \overline{T} := \sum_{(i,j)\in P_1} d_{ij}^g - 1 \ . \tag{22}$$

Proof: Since the value of a static flow $g^{\hat{T}} \in \mathcal{G}^{\hat{T}}$ is a subgradient of $v(T)$ at $T = \hat{T}$ (see Lemma 4), computing the interval $\mathcal{I}_{\hat{\Delta}}$ reduces to computing the interval of all values of T such that the min cost flow g of value $\hat{\Delta}$ belongs to the class $\mathcal{G}^T$. This is of course a classical problem from sensitivity analysis. Given the circulation $\tilde{g}$ obtained from the flow g by adding a return arc with flow $|g|$, we want to determine the smallest and the largest value of T, $\underline{T}$ and $\overline{T}$, respectively, such that $\tilde{g}$ is an optimal solution to problem $(MCCP)_T$.

Applying the classical negative cycle theorem (see e.g. Ahuja et al. [2]), yields that $\tilde{g}$ is optimal as long as the residual network $\mathcal{N}^{\tilde{g}}$ contains no negative cycle. Since $\mathcal{N}^g$ contains no negative cycle due to the optimality of g (the shortest paths P_1 and P_2 are thus well defined), a negative cycle C in $\mathcal{N}^{\tilde{g}}$ must contain either (s', s) or (s, s'), the only arcs not present in $\mathcal{N}^g$. Hence we have to distinguish the following two cases for the cycle C:

1. C consists of the arc (s', s) followed by a path P from s to s'. Since $d_{s's}^{\tilde{g}} = d_{s's} = -(T + 1)$, its cost $d(C)$ is then given by

$$d(C) := -(T + 1) + \sum_{(i,j)\in P} d_{ij}^g \ . \tag{23}$$

2. C consists of the arc (s, s') followed by a path P' from s' to s. Due to $d_{ss'}^{\tilde{g}} = -d_{ss'} = T + 1$ we have

$$d(C) := (T + 1) + \sum_{(i,j)\in P'} d_{ij}^g \ . \tag{24}$$

Requiring $d(C) \geq 0$ for the expressions (23) and (24) yields

$$T \leq \sum_{(i,j)\in P} d_{ij}^g - 1 \qquad \text{and} \qquad T \geq -\sum_{(i,j)\in P'} d_{ij}^g - 1 \ .$$

The most stringent restrictions on T result by using for P path P_1, the shortest path from s to s', and for P' path P_2, the shortest path from s' to s. □

The method described above not only computes the interval $\mathcal{I}_{\Delta_c}$, but also yields the line segment of $v(T)$ having slope Δ_c in case it exists. Let T^* be the value of T for which this line segment crosses the horizontal line L_v. In case $T^* \in \mathcal{I}_{\Delta_c}$ we are finished ($T(v) = \lceil T^* \rceil$). If however $T^* < \underline{T}$, the current slope Δ_c is obviously too large, while if $T^* > T$, it is too small. Let $c^g(P_r) := \min_{(i,j) \in P_r} c_{ij}^g$ denote the residual capacity of the paths P_r, $r = 1, 2$ defined in Lemma 6. Then in the first case ($T^* < \underline{T}$) we set $\Delta_u := \Delta_c - c^g(P_2)$ and in the latter case we set $\Delta_l := \Delta_c + c^g(P_1)$, and continue the binary search accordingly.

It is now easy to see that the time complexity of this algorithm is equal to $O(\min(\log v, \log |g_{max}|) \cdot MIN(n, m))$, where again $M(n, m)$ is the time needed to solve a minimum cost flow problem with n nodes and m arcs.

Unfortunately, in contrast to the binary search algorithm on the T-axis we did not succeed in developing an interpolation approach for the above binary search approach with respect to the slopes. The practical performance of the algorithm described above can be improved, however, if we also maintain an interval $[T_l, T_u]$ on the T-axis and use the subgradients at T_l and T_u for obtaining better bounds Δ_l and Δ_u. We will give some more details on this hybrid approach in the next section.

5 Numerical Investigations

5.1 Implementations

5.1.1 Implementation of the Polynomial Methods of Section 4

For solving the initial maximum flow problem and the shortest path problems which occur as subproblems in our search algorithms we used efficient implementations from literature, more specifically we applied the max flow code GOLDRMF of Derigs and Meier [13, 14] and the shortest path code L-2QUEUE of Gallo and Pallottino [16], respectively.

At this time, it appears that the relaxation algorithm of Bertsekas and Tseng [6] and the primal network simplex algorithm (see e.g. Grigoriadis [18]) are the two fastest algorithms for solving the minimum cost network flow problem. For our experiments we had available the code RELAXT of Bertsekas and Tseng [6] and the primal simplex code NET of Ahrens and Finke [1]. Observe that for evaluating $v(T)$ for given T, it is essential that for different values of T, the corresponding minimum cost circulation problems $(MCCP)_T$ differ only in the cost of the return arc which is $-(T + 1)$. Fortunately, both RELAXT and NET are well suited for performing reoptimization, i.e. starting from a previous optimal solution for a different T instead of recomputing from scratch.

In the final versions of our codes we applied NET, because in a first set of tests, NET turned out to be considerably faster on problems of type $(MCCP)_T$ than RELAXT, both when starting from scratch and when reoptimization was applied. The main reason for the bad behaviour of RELAXT seems to be that in instances of type $(MCCP)_T$ typically a small set of arcs, in particular the return arc, has much higher cost than the remaining arcs. (After completion of our study we learnt that now an improved relaxation code RELAXT-III with a possibly better performance on problems of this type is available (see Bertsekas and Tseng [7].)

Recall that for performing binary search on the slopes, an additional minimum cost flow problem has to be solved in order to determine for a given slope $\hat{\Delta}$ the associated optimality interval $\mathcal{I}_{\hat{\Delta}}$, where for different values of $\hat{\Delta}$ only the value of the flow to be computed changes. The computational results of Ali, Padman and Thiagaran [4] suggest that in such cases reoptimizing by an efficient implementation of the dual simplex method is preferable to introducing artificial high-cost arcs in the primal approach. As the relaxation method of Bertsekas and Tseng can also handle the case of supply/demand changes in a straightforward way, we are again left with the choice between two different methods for reoptimization.

5.1.2 Implementation of the Pseudopolynomial Methods of Section 4

The most natural way of implementing the parametric approach described in Section 4 would be to solve the parametric minimum cost flow problem $(MCCP)_T$ with a straightforward extension of the primal network simplex method (for the general idea of the parametric simplex method see Murty [31]). Whereas in the general case in each step of the algorithm all $O(m)$ non-basic arcs have to be checked for their optimality (dual feasibility), in our case it suffices to consider all arcs in the cocycle obtained by deleting the return arc from the basis since the return arc is the only arc with a parametric cost.

There is, however, also a dual approach. According to Lemma 6, to each segment of $v(T)$ of slope $\hat{\Delta}$ there corresponds a minimum cost static flow g of value $|g| = \hat{\Delta}$. (If g has cost $\sum_{(i,j)\in A} \tau_{ij} g_{ij}$ then the associated segment of $v(T)$ lies on the line $(T+1)|g| - \sum_{(i,j)\in A} \tau_{ij} g_{ij}$.) Considering the flow value as parameter leads to a flow problem with a parametric supply/demand vector which can be solved by an extension of the dual network simplex method. While for general parametric right-hand-side flow problems all $n-1$ basic arcs have to tested for their primal feasibility, in this special case it is enough to check only the arcs on the unique path in the basis tree from the source s to the sink s'. (On all other arcs the flow is independent of the parameter.)

So in principle, from a computational point of view the primal and the dual parametric approach are equivalent when appropriately implemented.

Unfortunately, no matter whether we start from a primal or a dual code, the necessary adaptions require a very detailed understanding of the nonparametric code to be modified. As we already had available both a primal and a dual parametric network simplex code from previous work (adaptions of NET of Ahrens et al. [1] and of MODAPT of Ali et al. [4], respectively), we thus decided not to undertake a new implementation which pays attention to specializations possible for our special problem in the computation of the optimality intervals. In this situation it is natural to choose the dual code since it needs only $O(n)$ instead of $O(m)$ time per computation of an optimality interval.

As the method based on *Minieka's* earliest arrival flow algorithm is essentially nothing else than solving a minimum cost flow problem by the shortest augmenting path method, its implementation is very easy. We simply used the shortest path code L-2QUEUE for successively obtaining a shortest path in the residual network and updated the residual network and the value of the current flow accordingly. Of course, there might exist more sophisticated implementations, but it was only our aim to get some feeling on the behaviour of this algorithm.

5.2 *Generation of Test Data*

Basically we used two classes of networks. The first class of instances was generated randomly by the code NETGEN of Klingman, Napier and Stutz [25]. For these instances it can be observed empirically that the number of breakpoints of $v(T)$ is linear in the number of arcs. In order to investigate the influence of the number of breakpoints on the algorithms to be tested, the second class was chosen to consist of the Zadeh based networks of Section 4 which yield an exponential number of breakpoints.

All NETGEN problems used in this study are networks with one source, one sink, no transshipment sources respective sinks, a cost range of 1–1000 and a capacity range of 50–5000. In order to achieve a comparatively large number of breakpoints, the total supply was always chosen as 10000 times the number of nodes. In this way we generated four types of networks, referred as N1–N4 in Table 1. These networks differ only in their number of nodes and arcs. For

Table 1. Characteristics of the networks used in our experiments

Problem Id.	*# nodes*	*# arcs*	*# breakpoints*	$\lvert g_{max}\rvert$
N1	200	2000	3638.3	2021992.2
N2	400	4000	10494.6	4015579.6
N3	400	8000	19121.6	4050916.8
N4	800	8000	28180.8	8017503.4
Z5	42	422	1048573.0	1310718.0

Table 2. v-dependent problem data for the networks from Table 1

Problem Id.	v	$T(v)$	$\Delta(T(v))$	$\#$ (*brps* $\leq T(v)$)
N1a	10^8	3956.2	66200.0	169.2
N1b	10^9	10380.2	209537.4	716.8
N1c	$5 \cdot 10^9$	22334.4	450820.0	1751.2
N1d	10^{10}	31688.8	605261.0	2336.2
N2a	10^7	1834.6	16122.0	34.4
N2b	10^9	9860.2	221523.0	960.0
N2c	10^{10}	30186.2	662796.4	3576.8
N2d	10^{11}	101851.2	1746391.2	9058.8
N3a	10^8	2640.0	87797.8	311.0
N3b	10^9	7431.2	285623.4	1373.8
N3c	10^{10}	22359.4	926580.6	4958.2
N3d	10^{11}	71267.6	2682658.2	13839.0
N4a	10^8	3789.6	69068.2	256.8
N4b	10^9	10088.4	212246.8	1169.6
N4c	10^{10}	31192.6	643996.6	4598.6
N4d	10^{11}	99020.6	1952731.8	14720.0
Z5a	10^7	3999.0	5000.0	4000.0
Z5b	10^{11}	399999.0	500000.0	400000.0

each type of network we created five instances and all computational results reported below are results averaged over five runs each. Table 1 furthermore provides the maximum flow value $|g_{max}|$ and the total number of breakpoints of $v(T)$ for the networks of types N1–N4.

Since all networks of the Zadeh type are of the same structure, we included only one particular instance from this class into our set of test problems. The network which is referred to as Z5 in Table 1 is obtained by setting $k := 20$. Note that this network leads to 1048573 breakpoints while having only 42 nodes.

We still need to fix the amount v of flow to be transmitted. In order to investigate the influence of the size of v on the difficulty of computing a quickest flow, we chose different values of v for our network types N1–N4 and Z5. In this way we obtained 22 different problem types. Table 2 contains for each of these types the value v, the minimum transmission time $T(v)$, the slope $\Delta(T(v))$ of the segment on which $T(v)$ lies, and finally in the last column, the number of breakpoints of the function $v(T)$ which are to the left of $T(v)$.

5.3 Computational Results

All our codes including the network flow subroutines we used are written in Fortran, but it should be mentioned that our codes have been set up for

experimental purposes only. A first set of experiments led to the following findings:

(1) For every single instance performing primal reoptimization from the very beginning outperforms clearly the variants where $v(T)$ is computed each time from scratch or where reoptimization starts only at a later stage.
(2) However, we found that for the other type of minimum cost flow problem where not the cost but the required flow value changes from iteration to iteration, the situation is far less favourable. Reoptimizing with the dual simplex algorithm from the very beginning turns out to be extremely inefficient – the resulting computation times are worse than those obtained by solving each problem from scratch by the primal simplex code. It seems to be very difficult, however, to find an appropriate criterion when to change over from the primal to the dual method. On the other hand, we found that the relaxation method of Bertsekas is better suited for performing reoptimization from the very beginning. Hence we used the method of Bertsekas in our final implementation of the binary search algorithm w.r.t. the slopes.
(3) With the exception of instances with a very small number of breakpoints to the left of $T(v)$ (about less than 50), the parametric method is much faster than the Minieka-based method. This is due to the fact that solving a shortest path problem is in general more expensive than performing a single pivot step.

Let us now have a closer look at the second stage of our computational study. For notational convenience we will use the following abbreviations in the tables given below: MTBIN, INTPL and NEWT denote the three search methods on the T-axis, namely binary search on the T-axis coupled with improved lower and upper bounds, the interpolation approach of Ibaraki, and Newton's approach. Further, ΔBIN denotes the binary search on the slopes and ΔHYBR is a hybrid method based on ΔBIN. Finally, DUAL stands for the (dual) parametric method.

In Table 3 below we report the computation times needed by the different methods for solving the problems of types N1, N2 and Z5. Table 4 contains some information on how many "basic steps" are performed by the various algorithms. For the search methods a basic step means one iteration, whereas for the parametric method it is more natural to treat one pivot as basic step.

The computation times displayed in Table 3 clearly indicate that contrary to what one would expect from the worst case analysis in Section 4, the search methods operating on the T-axis outperform the methods searching in the set of slopes. There are two main reasons for the rather disappointing behaviour of the methods ΔBIN and ΔHYBR. One is that the method we used for evaluating $v(T)$ for a given $T = \hat{T}$ is far more efficient than the method for computing the interval $\mathcal{I}_{\hat{\Delta}}$ for a given slope $\hat{\Delta}$. The other is that we did not succeed in finding for ΔBIN an equally effective means for accelerating the decrease of the search interval as

Table 3. CPU-seconds on a VAX 4300

Id.	MTBIN	INTPL	NEWT	ΔBIN	ΔHYBR	DUAL
N1a	6.026	5.982	6.274	9.296	10.158	2.110
N1b	6.666	6.470	6.740	12.416	12.666	5.260
N1c	6.270	6.214	6.558	14.742	11.710	10.946
N1d	6.488	6.024	6.198	15.534	12.462	14.586
N2a	15.500	14.588	15.636	18.742	20.536	4.230
N2b	33.856	31.606	35.904	55.014	60.688	19.652
N2c	34.488	31.822	34.306	64.466	60.644	53.368
N2d	34.462	34.480	33.074	98.668	68.842	134.028
Z5a	0.260	0.180	0.370	0.910	0.580	5.510
Z5b	0.270	0.210	0.270	1.390	0.750	554.180

Table 4. Number of basic steps

Id.	MTBIN	INTPL	NEWT	ΔBIN	ΔHYBR	DUAL
N1a	4.4	2.8	7.8	11.2	3.2	663.8
N1b	4.8	2.4	7.0	12.0	2.8	1501.4
N1c	5.4	2.6	6.0	11.4	2.0	2712.2
N1d	4.6	2.2	5.8	13.0	2.0	33351.4
N2a	4.6	3.4	8.4	10.6	3.0	650.2
N2b	6.0	2.4	8.0	14.2	3.0	2734.2
N2c	3.8	2.8	7.0	14.2	2.4	6264.8
N2d	3.6	2.2	4.6	13.4	1.8	12439.2
Z5a	7.0	1.0	12.0	19.0	5.0	5034.0
Z5b	5.0	1.0	6.0	18.0	2.0	500041.0

we have found for the T-search procedures. It turned out that the main idea underlying the hybrid approach ΔHYBR, namely to maintain besides the interval $[\Delta_l, \Delta_u]$ for the slope also an interval $[T_l, T_u]$ for $T(v)$ and to use the subgradients at T_l and T_u for improving the current slope bounds Δ_l and Δ_u, hardly pays off. Of course, on one hand the number of iterations can drastically be reduced by this approach (see Table 4), but on the other hand the computational effort per iteration increases significantly as for obtaining the subgradients at T_l and T_u, respectively, two additional min cost circulation problems have to be solved.

If we compare the T-search methods among themselves, it turns out that in most cases MTBIN and INTPL are slightly faster than NEWT for almost all problem instances, possibly except for those with a very large v. This is probably due to the choice of the initial lower bound T_l from (16) as starting value $T^{(0)}$ in the Newton iteration. It might well be the case that for smaller values of v, the

Table 5. Running times and number of basic steps of the most efficient algorithms found so far

Id.	MTBIN		INTPL		DUAL	
	# it.	*CPU*	*# it.*	*CPU*	*# pivots*	*CPU*
N3a	5.2	54.220	2.4	49.256	1580.0	17.884
N3b	5.8	75.596	2.4	72.116	3542.6	41.864
N3c	5.6	72.936	2.4	69.738	8856.2	119.53
N3d	2.8	83.406	2.8	78.182	20209.8	359.990
N4a	5.0	113.216	2.6	112.708	2440.0	34.206
N4b	5.6	190.184	2.0	186.970	4844.8	74.660
N4c	6.4	197.078	2.4	184.490	11146.6	185.006
N4d	5.4	189.628	2.4	192.404	14721.0	507.280

results of NEWT can be improved by using a different $T^{(0)}$, e.g. one can proceed as described in Remark 2 in the previous section.

Further, note that the number of iterations of INTPL is remarkably small. Due to the very uniform structure of $v(T)$ for the Zadeh-networks it is even equal to 1 for the problems Z5a and Z5b. However, in each iteration of INTPL a nonlinear equation has to be solved which leads again to an increased effort per iteration when compared to the simple binary search algorithm MTBIN.

Let us now compare the best search approaches, MTBIN and INTPL, respectively, with the parametric method DUAL. In order not to draw wrong conclusions, we additionally solved a set of larger quickest flow problems with these three approaches. Table 5 contains the results of these runs.

It is easy to see that the behaviour of DUAL strongly depends on the number of breakpoints of $v(T)$ which are $\leq T(v)$, while this number has almost no effect on the two other methods. Note that independent of the problem size the number of iterations in MTBIN and INTPL is almost constant. But DUAL gets worse and worse the larger the number of breakpoints becomes. This effect is demonstrated particularly well by the increase in the CPU-time from the Zadeh-example Z5a to the Zadeh-example Z5b in Table 3.

As far as the almost constant number of iterations needed by the search approaches of Section 4 is regarded, we have confirmed the findings which Ibaraki obtained in [21] for other types of fractional programming problems. However, our problems, in particular the rather small Zadeh-examples, seem to be more difficult to solve as we need more iterations per problem than Ibaraki.

Let us now try to answer the question which of the algorithms mentioned above is best suited for solving a given practical *(QFP)*. Generally speaking, one can say that as long as the number of breakpoints to the left of $T(v)$ is not too large, algorithm DUAL is the winner, whereas otherwise it is recommended to use the methods MTBIN or INTPL. (See e.g. the change in the behaviour that occurs between $v = 10^9$ and $v = 10^{10}$ for the NETGEN-example N3.)

Obviously, the algorithm DUAL can be modified such that, depending on the size of v, it starts alternatively from the zero flow proceeding then from left to

right as it is done in our implementation or from a maximum flow proceeding from right to left. This will yield an improvement for cases where only a small number of breakpoints lies to the right of $T(v)$. But note that in example N3c this is not the case. There the number of breakpoints to the left of $T(v)$ is according to Table 2 approximately equal to 4958, which is only about a quarter of the total number of breakpoints of $v(T)$ for this instance (confer Table 1).

A closer investigation revealed that the main reason why DUAL is superior for not too large values of v is that in this approach only one almost trivial minimum cost flow problem (for the flow value 0) has to be solved from scratch while in the search procedures such as MTBIN and INTPL we have to solve two minimum cost flow problems, one for $T = T_l$ and one for $T = T_u$ at the beginning. Especially for the large NETGEN-example it turned out that evaluating $v(T)$ at the initial T_u is very expensive. (These problems get more and more difficult, the larger T becomes.)

To conclude this section, let us point out that our primary objective in implementing the search methods of Section 4 was getting a low number of iterations and only the secondary objective was to minimize the CPU-times. During our investigations it turned out, however, that in many cases, the running times decrease if we use weaker but easier computable lower and upper bounds T_l and T_u, while usually as a consequence thereof the number of iterations increases.

6 A Strongly Polynomial Algorithm

In this section we will propose a strongly polynomial algorithm for the *(QFP)* which relies on an ingenious idea of Megiddo [28] for solving linear fractional programs. Throughout this section we will assume familiarity with Megiddo's work. (A self-contained description can be found in Burkard, Dlaska and Klinz [9].)

Basically our aim is to solve the problem $(MCCP)_T$ for $T = T(v)$, but unfortunately we do not know $T(v)$. Following Megiddo we can however proceed as follows: We extend a strongly polynomial algorithm A for the minimum cost circulation problem to the problem with parametric costs by extending the operations of A from the set of reals to the set of affine-linear functions. For this approach to work, the only operations on the arc costs algorithm A is allowed to perform, are additions and subtractions of two costs, multiplications of an arc cost with a real number and finally pairwise comparisons between arc costs. Currently the best algorithm fulfilling these requirements has a time complexity of $O(m \log n(m + n \log n))$ and is due to Orlin [32].

Let J be an initial interval such that $T(v) \in J$. Megiddo's method now mainly relies on the fact that as long as no comparisons involving the arc costs occur,

all steps of the algorithm can be performed immediately without knowing $T(v)$. If we arrive at a comparison between two parametric arc costs, we first determine the critical value T^* for which the two cost functions intersect. In case that T^* does not exist or lies outside of J, the result of the comparison is obviously independent of $T(v)$ and can be determined immediately. Otherwise, we compute the function value $v(\lfloor T^* \rfloor)$, e.g. by calling algorithm A as subroutine. The result of this computation enables us to decide on which side of T^* the value $T(v)$ lies and to update J. This process eventually will yield a circulation g such that g is optimal over J and $T(v) \in J$. Then $T(v)$ can be obtained by a straightforward computation.

It is easy to see that this algorithm determines the minimum transmission time $T(v)$ in $O(m^2 \log^2 n(m + n \log n)^2)$ time. (There are at most $O(m \log n(m + n \log n))$ comparisons in Orlin's min cost flow algorithm [32], and each comparison causes at most one call of Orlin's algorithm as subroutine.) Hence a strongly polynomial time algorithm for the *(QFP)* running in $O(m^2 \log^2 n(m + n \log n)^2)$ time follows.

It is worth-mentioning that the algorithmic scheme described above can be sped up provided one succeeds in reducing the number of calls to algorithm A. In [28, 29] Megiddo suggested several ways of achieving this aim for various types of fractional programming problems.

It is not very difficult to derive from Megiddo's approach which is based on exploiting parallelism an $O(m^2 \log^5 n(m + n \log n))$ time algorithm for solving the *(QFP)*. (There we apply a parallel variant of Orlin's min cost flow algorithm which performs $O(m \log n)$ shortest path computations. Each of these shortest path problems can be solved in $O(\log^2 n)$ time on a parallel machine with $O(n^3/\log n)$ processors.) Further details can be found in Megiddo [29] and in Burkard et al. [9].

An even better strongly polynomial algorithm for the *(QFP)* can be obtained by observing that in the parametric minimum cost circulation problem $(MCCP)_T$ the return arc is the only arc that has a parametric cost. Suppose we apply Megiddo's original algorithm with Orlin's min cost flow algorithm as underlying routine. The essential idea is now that due to the special property of the costs, the shortest path problems that occur as subproblems in Orlin's min cost flow algorithm can be solved directly as special parametric shortest path problems without applying the scheme of Megiddo. A careful analysis of the results of Young, Tarjan and Orlin [41] and Karp and Orlin [24] on special parametric shortest path problems shows that in the case where all arcs except one have constant costs, the associated optimal value function has at most n breakpoints. Hence these problems are solvable asymptotically in the same time as already needed for solving a single shortest path problem. (This has been observed independently e.g. by Orlin [33] and Klinz and Tuy [26].) Therefore, each of the $O(m \log n)$ shortest path computations necessary in Orlin's algorithm results in a number of at most $n - 1$ critical values $T_1^* \leq T_2^* \leq \cdots \leq T_r^*$, $r < n$. Using at most $O(\log n)$ calls to Orlin's algorithm [32] one can then determine the interval $[T_p^*, T_{p+1}^*]$ with $T(v) \in [T_p^*, T_{p+1}^*]$. Along these lines an $O(m^2 \log^3 n(m + n \log n))$

time algorithm for the *(QFP)* can be developed. (This algorithm was suggested for the first time by Orlin [33] for solving a constrained maximum flow problem, see also Ahuja and Orlin [3].)

Whereas Megiddo's method is apparently very effective from the theoretical point of view, nothing seems to be known about its practical efficiency. Many researchers suspect that due to the relatively large overhead necessary for making the algorithm strongly polynomial, Megiddo-based algorithms will be inferior to its just polynomial competitors for solving practical problems.

In the case of the *(QFP)* the polynomial methods of Section 4 are superior to the Megiddo-based approach even from the point of view of worst case complexity for all instances of reasonable size, i.e. more specifically if $\log \Delta_{max}$ is $o(m \log^2 n)$, where $\Delta_{max} = \min\{v, |g_{max}|\}$. Thus it seemed to us that it does not pay off to undertake the rather cumbersome and time-consuming task of implementing Megiddo's method.

7 Summary and Concluding Remarks

In this paper we investigated a special dynamic flow problem, the quickest flow problem *(QFP)*. This problem in a certain sense is the inverse problem of the classical maximum dynamic flow problem. Moreover, it turned out that the *(QFP)* can be written as linear fractional program.

For a network with n nodes, m arcs and maximum static flow value $|g_{max}|$ our best polynomial algorithm for solving the *(QFP)* for a given value v runs in $O(m \log n(m + n \log n) \log \Delta_{max})$ time where $\Delta_{max} = \min\{v, |g_{max}|\}$. Furthermore, a strongly polynomial algorithm of time complexity $O(m^2 \log^3 n(m + n \log n))$ can be obtained.

Finally, we conducted a computational study on the practical behaviour of the various methods for solving the *(QFP)*. The results obtained seem to provide strong evidence that the best of the tested algorithms may well serve as an efficient tool for solving large practical problems.

Despite the results on quickest flows presented in this paper and in the paper by Burkard, Dlaska and Kellerer [8], there are still several open questions which might deserve further research. Let us very briefly mention some of them.

(1) In the context of evacuation models the following generalization of the quickest flow problem to the case of more than one source occurs (see Hamacher [19]): Given a network $\mathcal{N}$ with $q \geq 1$ sources $s_1, \ldots, s_q$ along with a number v_{s_k} for each source s_k, and one sink s', determine the minimum number of time units that are necessary to send v_{s_k} units of flow from the source s_k to the sink s', for $k = 1, \ldots, q$. To our knowledge no efficient algorithm for this problem is known.

(2) The perhaps most challenging research question related to quickest flows is to find a polynomial algorithm for the quickest flow problem with a better worst case complexity than the algorithms presented in our paper. Currently we are investigating whether some of the ideas which have proven successful for the minimum mean cycle problem (see Orlin and Ahuja [34]) and the maximum mean cut problem (see Iwano et al. [22]) can also be applied in our case.

(3) Since the computational results of the previous section have shown that the number of iterations is on the average much smaller than predicted by our worst case bounds, the twofold question of finding tight worst case bounds and associated worst case examples arises. Such worst case examples would be of particular interest for the Newton approach, for the binary search algorithm with improved lower and upper bounds and finally above all for the interpolation algorithm.

(4) In all our methods we have evaluated $v(T)$ exactly. It is an interesting question from the point of view of both theory and practice whether computing only some kind of approximate min cost flow can help to improve the theoretical and/or practical efficiency of the search methods described in this paper.

Acknowledgement: We would like to thank Prof. Agha I. Ali for kindly making his code MODAPT available to us. Moreover, we are grateful to Günter Rote for valuable discussions that lead to an improvement of a first draft of this paper, and to Prof. James B. Orlin for a discussion on the strongly polynomial algorithm mentioned in Section 6.

References

[1] Ahrens JH, Finke G (1980) Primal transportation and transshipment algorithms. *Zeitschrift für Operations Research* 24:1–32

[2] Ahuja RK, Magnanti T, Orlin JB (1989) Network Flows. In: GL Nemhauser et al. (eds.), *Handbooks in OR & MS*, Vol. 1, North Holland, Amsterdam, 211–369

[3] Ahuja RK, Orlin JB (1991) Scaling algorithms for the constrained maximum flow problem, talk presented at the 14-th International Symposium on Mathematical Programming, Amsterdam

[4] Ali AI, Padman R, Thiagaran H (1989) Dual algorithms for pure network problems. *Operations Research* 37:159–171

[5] Aronson JE (1989) A survey on dynamic network flows. *Annals of Operations Research* 20:1–66

[6] Bertsekas DP, Tseng P (1988) The relax codes for linear minimum cost network flow problems. *Annals of Operations Research* 13:125–190

[7] Bertsekas DP, Tseng P (1990) RELAXT-III: A new and improved version of the RELAX code, LIDS Report P-1990, Laboratory for Information and Decision Systems, MIT, Cambridge, MA

[8] Burkard RE, Dlaska K, Kellerer H (1991) The quickest disjoint flow problem. Technical Report 189-91, Institute of Mathematics, University of Technology, Graz, Austria

[9] Burkard RE, Dlaska K, Klinz B (1991) The quickest flow problem. Technical Report 188-91, Institute of Mathematics, University of Technology, Graz, Austria (also available as Rutcor Research Report RRR # 57-91, Rutgers University, New Brunswick, NJ, 1991)
[10] Carstensen PJ (1983) Complexity of some parametric integer and network programming problems. *Mathematical Programming* 5:64–75
[11] Chalmet LG, Francis RL, Saunders PB (1982) Network models for building evacuation. *Management Science* 28:86–105
[12] Chen YL, Chin YH (1990) The quickest path problem. *Computers and Operations Research* 17:153–161
[13] Derigs U, Meier W (1989) Implementing Goldberg's max-flow algorithm, a computational investigation. *Zeitschrift für Operations Research* 33:383–403
[14] Derigs U, Meier W (1990) Goldrmf/Goldnet-max-flow program. *European Journal of Operational Research* 46:260
[15] Ford LR, Fulkerson DR (1962) *Flows in Networks*, Princeton University Press, New Jersey
[16] Gallo G, Pallottino S (1988) Shortest path algorithms. *Annals of Operations Research* 13:3–79
[17] Gibbons A, Rytter W (1988) *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge
[18] Grigoriadis MD (1986) An efficient implementation of the network simplex method. *Mathematical Programming Study* 26:83–111
[19] Hamacher HW (1983) Min cost and time minimizing dynamic flows, Research Report No. 83-16, Industrial & Systems Engineering Department, University of Florida, Gainesville
[20] Hamacher HW (1989) Temporally repeated flow algorithms for dynamic min cost flows, *Proceedings of the 28-th IEEE Conference on Decision and Control*
[21] Ibaraki T (1983) Parametric approaches to fractional programs. *Mathematical Programming* 26:345–362
[22] Iwano K, Misono S, Tezuka S, Fujishige S (1990) A new scaling algorithm for the maximum mean cut problem. IBM Research Report RT 0049, Tokyo, Japan
[23] Jarvis JR, Ratliff DH (1982) Some equivalent objectives for dynamic network flow problems. *Management Science* 28:106–109
[24] Karp RM, Orlin JB (1981) Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics* 3:37–45
[25] Klingman D, Napier A, Stutz S (1974) Netgen: A program for generating large scale capacitated assignment, transportation and minimum cost flow problems. *Management Science* 20: 814–821
[26] Klinz B, Tuy H (1991) Minimum concave-cost network flow problems with a single nonlinear arc cost, Technical Report 191-91, Institute of Mathematics, University of Technology, Graz, Austria
[27] McCormick TS, Ervolina TR (1990) Computing maximum mean cuts, UBC Faculty of Commerce Working Paper 90-MSC-011, Vancouver, BC
[28] Megiddo N (1979) Combinatorial optimization with rational objective functions. *Mathematics of Operations Research* 4:414–424
[29] Megiddo N (1983) Applying parallel computation algorithms in the design of serial algorithms. *Journal of the A. C. M.* 30:852–865
[30] Minieka E (1973) Maximal, lexicographic, and dynamic network flows. *Operations Research* 21:517–527
[31] Murty KG (1983) *Linear Programming*, John Wiley & Sons, New York
[32] Orlin JB (1988) A faster strongly polynomial minimum cost flow algorithm, *Proc. 20-th Annual Symp. Theory of Computing* 377–387
[33] Orlin JB (1992) Private communication
[34] Orlin JB, Ahuja RK (1992) New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming* 54:41–56
[35] Radzik T (1991) Minimizing capacity violations in a transshipment network, Technical Report, Computer Science Department, Stanford University, CA

[36] Rosen JB, Sun S-Z, Xue G-L (1991) Algorithms for the quickest path problem and the enumeration of quickest paths. *Computers and Operations Research* 18:579–584
[37] Rote G (1991) Private communication
[38] Ruhe G (1991) *Algorithmic Aspects of Flows in Networks*, Mathematics and Its Applications, Volume 69, Kluwer Academic Publishers, Doortrecht
[39] Schaible S (1978) *Analyse und Anwendungen von Quotientenprogrammen – Ein Beitrag zur Planung mit Hilfe der nichtlinearen Programmierung*, (in German), Mathematical Systems in Economics 42, Verlag Anton Hain, Meisenheim am Glan
[40] Schaible S, Ibaraki T (1983) Fractional programming. *European Journal of Operational Research* 12:325–338
[41] Young NE, Tarjan RE, Orlin JB (1991) Faster parametric shortest path minimum-balance algorithms. *Networks* 21:205–221
[42] Zadeh N (1973) A bad network problem for the simplex method and other minimum cost flow algorithms. *Mathematical Programming* 5:255–266

Received: November 1991
Revised version received: April 1992