Max-min fair distribution of modular network flows on fixed paths

Pål Nilsson[1] and Michał Pióro[12]

[1] Department of Communication Systems, Lund University, Sweden,
Box 118 SE-221 00, Lund, Sweden
{paln,mpp}@telecom.lth.se

[2] Institute of Telecommunications, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warszawa, Poland

Abstract. In this paper a new aspect of the classical max-min fairness fixed-path problem is investigated. The considered (multi-criteria) optimization problem is almost identical to the continuous-flow problem, with the additional complicating assumption that flows must be integral. We show that such an extension makes the problem substantially more difficult (in fact $\mathcal{NP}$-hard). Through comparison with the closely related continuous-flow problem, a number of properties for the solution of the extended problem are derived. An algorithm, based on sequential resolution of linear programs, that shows to be useful (produce optimal solutions) for many instances of the considered problem, is given. It follows that this algorithm can be made exact, through substituting the involved linear programs by mixed-integer programs.

Keywords: Max-min fairness, Network optimization, Modular flows

1 Introduction

This paper concerns Max-Min Fair (MMF) allocation of bandwidth to demands (users) in a communication network. The MMF allocation principle is often considered in the context of IP networks carrying elastic traffic. It is also relevant when several local networks are to be connected via an overlay network with given overlay link capacities. In such a situation it has to be decided how much bandwidth to assign to the pairs of these local networks.

We consider the case when one (fixed) path per demand is used, and when demands can be assigned bandwidth only in multiples of a predefined module. This is a practical extension of the frequently cited MMF problem addressed in [1]. The modular (integral) flow requirement has to the best our knowledge not been studied in this context before. However, a great deal of work has been carried out considering other versions of this problem. In [1] it is shown how MMF is obtained if paths are fixed and demand flows are continuous. It is also well-known how MMF can be achieved if flows are allowed to split over several paths and demand flows are continuous [2–4]. The hardness of computing MMF continuous flows, forcing single-path selection, was pointed out in [5, 6].

Integral flow volumes, certainly being the source of difficulty for the problem considered in this paper, can be well motivated from a practical viewpoint. This requirement models that each demand volume must be a multiple of a predefined module, and is a consequence of that in a real network there is always a smallest trading unit, prohibiting flows from being continuous. It will be assumed that a problem instance is given as a network with link capacities, a set of source-destination node-pairs (S-D pairs), where each S-D pair represents a requirement for bandwidth (demand), and a path for each demand. For such an instance, the following traffic engineering problem is addressed: the demand between each S-D pair must be assigned a modular flow volume, such that the sum of flows on each link does not violate the link capacity. The distribution of flows among the S-D pairs must obey the MMF principle.

1.1 Notation

Throughout the paper we will use the following notation. Vertices (nodes) are labeled with index v, where $v = 1, 2, \ldots, V$, and V is the number of vertices. Vertices are interconnected by edges (links) labeled with index e, where $e = 1, 2, \ldots, E$, and E is the number of edges. Edges are assumed to be undirected. Each edge e is assigned a certain given capacity denoted by c_e. Demands are labeled with index d, where $d = 1, 2, \ldots, D$, and D is the number of demands. A demand is a requirement for bandwidth between a vertex-pair in the network. Each demand d is assumed to be associated with one selected simple path. A path is a set of edges that connects a vertex-pair. A binary indicator, δ_{ed}, is used for the edge-demand incidence relation: $\delta_{ed} = 1$ if edge e belongs to the path of demand d, and $\delta_{ed} = 0$ otherwise. The total flow allocated to demand d (on its corresponding path) will be identified by x_d. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$ denote the vector where entry number d is the flow x_d allocated to demand d (also called the allocation vector), and let $\vec{\boldsymbol{x}}$ be the allocation vector sorted in non-decreasing order. The sorted allocation vector, $\vec{\boldsymbol{x}}$, is said to be lexicographically greater than the sorted allocation vector, $\vec{\boldsymbol{x}}'$, $\vec{\boldsymbol{x}} \succ \vec{\boldsymbol{x}}'$, if the first non-zero entry of $\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}'$ is positive. Consequently, $\vec{\boldsymbol{x}} \succeq \vec{\boldsymbol{x}}'$ means $\vec{\boldsymbol{x}} \succ \vec{\boldsymbol{x}}'$ or $\vec{\boldsymbol{x}} = \vec{\boldsymbol{x}}'$.

1.2 Problem Description

This study concerns the problem of assigning modular flows to demands in a capacitated network, such that the distribution of flows among demands is MMF. The MMF principle is to first assure that the demand that gets the least flow gets as much as possible, then that the demand that gets the second least flow gets as much as possible, and so on. Formally, in an MMF allocation of flows each demand is assigned a flow such that it holds for the sorted allocation vector that an entry can be increased only at the cost of decreasing a previous entry, or by making the allocation vector infeasible[3]. It can be shown that obtaining

[3] Since this is a property of the sorted allocation vector entries and not for the specific demands, the definition is valid even for non-convex versions of the problem.

an allocation vector, $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, with this characteristic is equivalent to solving

$$\text{lex max } \vec{\boldsymbol{x}};\ \boldsymbol{x} \in Q, \tag{1}$$

where Q is the set of feasible solutions [4]. In other words, (1) seeks for an allocation vector, $\boldsymbol{x}^* \in Q$, for which it holds that $\vec{\boldsymbol{x}}^* \succeq \vec{\boldsymbol{x}}$ for all $\boldsymbol{x} \in Q$. In this paper, the set of feasible solutions Q is defined by the following two requirements:

(i) $\sum_d \delta_{ed} x_d \leq c_e$, $e = 1, 2, \ldots, E$, and
(ii) $x_d \in \mathbb{Z}^+$ for all demands $d = 1, 2, \ldots, D$,

where $\mathbb{Z}^+$ is the non-negative integers. The above two constraints mean in turn that the sum of flows on an edge cannot exceed the edge's capacity, and that each flow must assume a non-negative integer. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, be a feasible solution to (1), with Q constituted only by (i), and let $\boldsymbol{x}^z = (x_1^z, x_2^z, \ldots, x_D^z)$, be a feasible solution to (1), with Q constituted by (i) and (ii). To simplify notation, let $\boldsymbol{y} = (y_1, y_2, \ldots, y_D) = \vec{\boldsymbol{x}}$ and $\boldsymbol{y}^z = (y_1^z, y_2^z, \ldots, y_D^z) = \vec{\boldsymbol{x}}^z$. We will denote optimal $\boldsymbol{x}$ an Optimal Continuous Solution (OCS), and optimal $\boldsymbol{x}^z$ an Optimal Integral Solution (OIS). Note that solving for OCS is precisely what is addressed in [1], and is well known to be accomplished by a simple algorithm (called "lifting" or "waterfilling") of polynomial time [1, 4].

Example 1. Consider the network given in Figure 1.2. One demand between each vertex-pair is assumed. Paths are evident. Edge capacities are given in the figure. The sorted OCS is $\boldsymbol{y} = (0.5, 0.5, 0.5)$, whereas the sorted OIS is $\boldsymbol{y}^z = (0, 1, 1)$.

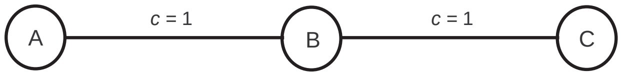


Fig. 1. A simple instance.

1.3 The assumption of modular flows

In practice, an edge cannot have an arbitrary capacity, but is installed in multiples of a predefined module, M. Moreover, it is reasonable to assume that for each demand d, $d = 1, 2, \ldots, d$, the demand's flow, x_d, must be a multiple of the same module. Without loss of generality we may, just changing units, assume that $M = 1$, and thus that $c_e \in \mathbb{Z}^+ \setminus \{0\}$, and, as is accomplished by (ii), require that $x_d \in \mathbb{Z}^+$ for all demands $d = 1, 2, \ldots, D$. Hence, modular flows can be treated as integral flows.

2 Some properties of the optimal integral solution

As the properties of the optimal continuous solution are very well known and understood [1, 3, 4], we will in this section address characterization of the optimal integral solution by comparing it to the optimal continuous solution. We assume that $\boldsymbol{y} = \vec{\boldsymbol{x}}$ is the sorted OCS, and $\boldsymbol{y}^z = \vec{\boldsymbol{x}}^z$ is the sorted OIS.

Property 1. If $\boldsymbol{y}^z \neq \boldsymbol{y}$ then there exists an entry k for which $y^z_k > y_k$.

Proof. It is easy to see that $\boldsymbol{y} \neq \boldsymbol{y}^z$ implies that there exists a demand d such that $x_d - \lfloor x_d \rfloor > 0$, because if $x_d \in \mathbb{Z}^+$, for all $d = 1, 2, \ldots, D$, then the OCS would be an OIS and necessarily $\boldsymbol{y} = \boldsymbol{y}^z$. Without loss of generality we may assume that $\boldsymbol{x} = \boldsymbol{y}$ (this may be obtained by alternative enumeration of demands). Consider the non-integral entries $x_i, x_{i+1}, \ldots, x_{i+m}$ for which it holds that for all entries k, $k < i$, (if any) $x_k \in \mathbb{Z}^+$, and if there exists an entry $i+m+1$ then $x_{i+m+1} \in \mathbb{Z}^+$. Construct a solution $\boldsymbol{x}^a$ by truncating all demand volumes of $\boldsymbol{x}$, except demand $i + m$, which is rounded up. The idea is illustrated in Figure 2. This solution

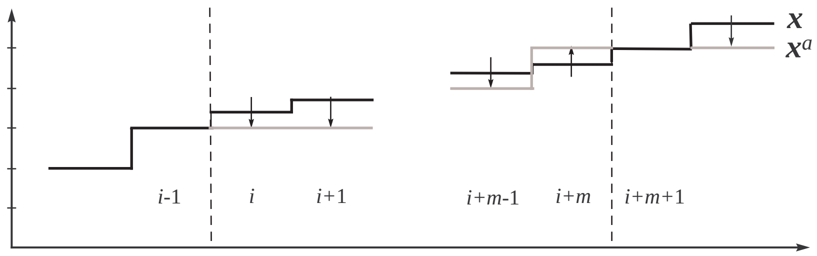


Fig. 2. How to obtain $\boldsymbol{x}^a$ from $\boldsymbol{x}$.

must be feasible since edge capacities are integral. Moreover $\boldsymbol{x}^a$ is an integral solution with the property that $\boldsymbol{x}^a = \vec{\boldsymbol{x}}^a (= \boldsymbol{y}^a)$. As it must hold that $\boldsymbol{y}^z \succeq \boldsymbol{y}^a$, we must have that either $y^z_j > y^a_j$ for some j, $i \leq j < m$, or that $y^z_j = y^a_j$ for all j, $i \leq j < m$ and that $y^z_{i+m} \geq y^a_{i+m}$. Both cases imply that there exists an index j, $i \leq j \leq i + m$, such that $y^z_j > y_j$. □

Property 2. If for some entry k, $y^z_k > y_k$, then there exists a demand d, such that $x^z_d > x_d$.

Proof. Without loss of generality assume that $\boldsymbol{x} = \boldsymbol{y}$ (if this is not true, reenumerate the demands). Suppose $\boldsymbol{x} \geq \boldsymbol{x}^z$ and consider all entries m and n, $1 \leq m < n \leq D$, such that $x^z_n < x^z_m$. Interchanging all such elements in $\boldsymbol{x}^z$, we will eventually arrive at $\boldsymbol{y}^z$. However, we have that $x_m \geq x^z_m > x^z_n$ and $x_n \geq x_m \geq x^z_m$, so it must be true that $\boldsymbol{x} = \boldsymbol{y} \geq \boldsymbol{y}^z$, which is a contradiction. □

Property 3. Let j be the largest integer for which it is true that $y_k - y_k^z \geq 0$, $1 \leq k \leq j$. Then, $j \geq 1$ and it holds that $y_k - y_k^z < 1$, for $1 \leq k \leq j$.

Proof. By definition such an entry j must exist. Suppose that for some k, $1 \leq k \leq j$, $y_k - y_k^z \geq 1$. Then we can find a feasible integral solution by just truncating the OCS. Call this solution $\boldsymbol{y}^t$. Apparently, $\boldsymbol{y}^t \succ \boldsymbol{y}^z$, which is a contradiction proving the second part of the statement. □

The following property is a direct analogy to a very well known property of the OCS [1, 4].

Property 4. For each demand d', there exists at least one saturated edge e for which $x_{d'}^z \geq \max_d\{x_d^z : \delta_{ed} = 1\} - 1$.

Proof. It is obvious that it is possible to find at least one saturated edge for each demand, since otherwise that demand could be increased. Denote by

$$\hat{x}_e^z = \max_d \{x_d^z : \delta_{ed} = 1\},$$

the flow of the maximal demand on edge e, and suppose, contradictory to the statement, that it holds for all such saturated edges e, for demand d', that $x_{d'}^z < \hat{x}_e^z - 1$. Then, for these saturated edges we have $\hat{x}_e^z > x_{d'}^z + 1$. Thus for each of the edges with this property, reassigning the currently maximal flow a value of $\hat{x}_e^z - 1$ and demand d' a value of $x_{d'}^z + 1$ give a lexicographically larger solution, which is a contradiction. □

It should be noted that Property 4 implies that for each demand d', if there does not exist a saturated edge for which $x_{d'}^z = \max_d\{x_d^z : \delta_{ed} = 1\}$, then there must exist a saturated edge for which $x_{d'}^z = \max_d\{x_d^z : \delta_{ed} = 1\} - 1$.

Property 5. For a feasible allocation vector $\boldsymbol{x}^m$, $\boldsymbol{x}^m \in (\mathbb{Z}^+)^D$, if it holds for each demand d', $d' = 1, 2, \ldots, D$, that $x_{d'}^m = \max_d\{x_d^m : \delta_{ed} = 1\}$ on at least one saturated link e for which $\delta_{ed'} = 1$, then $\boldsymbol{x}^m$ is the unique OIS.

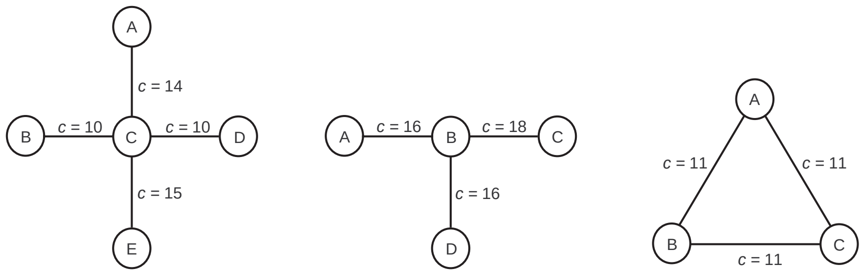
Proof. The result is valid for the continuous flows case [4], i.e., a feasible allocation vector, $\boldsymbol{x}$, for which it holds that for each demand d', $x_{d'} = \max_d\{x_d : \delta_{ed} = 1\}$ on at least one saturated link e for which $\delta_{ed'} = 1$, is the unique OCS. Now since $\boldsymbol{x}^m \in (\mathbb{R}^+)^D$, $\boldsymbol{x}^m$ is the unique OCS and therefore the unique OIS. □

The following examples illustrate that it may happen, considering the OCS and the OIS for a given instance, that $x_d^z < \lfloor x_d \rfloor$ and that $x_d^z > \lceil x_d \rceil$. They also show that there is no certain throughput domination, i.e., there exist both instances for which $\sum_d x_d^z < \sum_d x_d$, and instances for which $\sum_d x_d^z > \sum_d x_d$.

Example 2. Consider the network given in Figure 3(a). There are 2 demands between verticess A and B, 2 between A and E, 2 between A and D, 1 between C and B, 1 between C and E, and 1 between C and D. The sorted OCS is $\boldsymbol{y} = (\underbrace{7/3, \ldots, 7/3}_{6}, 16/3, 16/3, 31/3)$ and the sorted OIS is $\boldsymbol{y}^z = (2, 2, 2, 2, 3, 3, 6, 6, 9)$.

Example 3. Consider the network given in Figure 3(b). There are 4 demands between verticess A and B, 2 between A and C, 4 between D and B, 2 between D and C, and 1 between D and B. The sorted OCS is $\boldsymbol{y} = (\underbrace{8/3, \ldots, 8/3}_{12}, 22/3)$ and the sorted OIS is $\boldsymbol{y}^z = (2, 2, 2, 2, \underbrace{3, \ldots, 3}_{8}, 10)$.

Example 4. Consider the network given in Figure 3(c). Edge capacities are given in the figure. There is one demand between each vertex-pair. Each demand is using the associated simple two-edge path. The sorted OCS is $\boldsymbol{y} = (5.5, 5.5, 5.5)$ and the sorted OIS is $\boldsymbol{y}^z = (5, 5, 6)$.



(a) Example 2. (b) Example 3. (c) Example 4.

Fig. 3. Example instances comparing the OCS and the OIS.

3 Computational complexity

In this section it will be shown that the problem studied in this paper is $\mathcal{NP}$-hard. This means that it is unrealistic to aim for a general polynomial time algorithm that obtains an MMF flow distribution, when integer flows on fixed paths are required. Mind that the considered optimization problem is exactly that considered in [1], but with integer-valued flows. As can be verified in Examples 2 and 3, the basic “lifting” algorithm (sometimes called “waterfilling”) presented in [1] is in general not applicable for the integer flows case. An attempt to use this basic procedure will show that certain non-trivial, discrete decisions occasionaly have to be taken. So there are certainly reasons to conjecture that this multi-criteria optimization problem is computationally hard. We will call the associated decision problem FIXED PATHS MMF – MODULAR FLOWS (FIXMMF-MF):

FIXMMF-MF:
INSTANCE: An edge capacity $c_e \in \mathbb{Z}^+$ for each edge $e = 1, 2, \ldots, E$, a binary edge-demand incidence coefficient, δ_{ed}, for each demand $d = 1, 2, \ldots, D$, and a target vector $\boldsymbol{x}^T \in (\mathbb{Z}^+)^D$.
QUESTION: Is there an assignment of flow, $x_d \in \mathbb{Z}^+$, for each demand d, such that $\sum_d \delta_{ed} x_d \leq c_e$ for each edge e, and such that if $\boldsymbol{x} = (x_1, x_2, \ldots, x_D)$, then $\vec{\boldsymbol{x}} \succeq \vec{\boldsymbol{x}}^T$?

Proposition 1. *FIXMMF-MF is $\mathcal{NP}$-complete.*

Proof. A nondeterministic algorithm needs only to guess an integral flow for each demand and check if the edges have the required capacity and if it holds for the resulting allocation vector, $\boldsymbol{x}$ that $\vec{\boldsymbol{x}} \succeq \vec{\boldsymbol{x}}^T$. Thus clearly, FIXMMF-MF is in $\mathcal{NP}$. We will transform the decision problem of SET PACKING into an instance of FIXMMF-MF. It is trivial to verify $\mathcal{NP}$-completeness of the former, restricting it to EXACT COVER BY 3-SETS, shown to be $\mathcal{NP}$-complete in [7].

SET PACKING:
INSTANCE: A collection of C finite sets and a positive integer $K \leq |C|$.
QUESTION: Does C contain at least K mutually disjoint sets?

Consider an arbitrary instance of SET PACKING. A collection C of n finite sets is given, $C = \{A_1, A_2, \ldots, A_n\}$. We will let each set A_i constitute a demand and the elements of each such set a chain of edges that is a path for that demand. Assume that there is N distinct elements in total in all of the sets A_i. For each such element a_k, $k = 1, 2, \ldots, N$, construct two vertices connected by one edge as in Figure 4. These edges will be referred to as the element edges. For each set

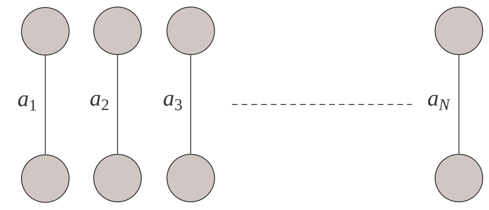


Fig. 4. Element edges.

$A_i \in C$ perform the following operations. Construct a source vertex s_i and a sink vertex t_i. Label the elements of set A_i such that $A_i = \{z_1, z_2, \ldots, z_m\}$, and note that there is one-to-one correspondence between these labeled elements and m of the element edges. Denote the upper vertex of the element edge corresponding to z_j, $j = 1, 2, \ldots, m$ by v_j^u, and the lower vertex by v_j^l. Add new edges connecting

s_i to v_1^u, v_1^l to v_2^u, v_2^l to v_3^u, and so on. Finally, add an edge that connects v_m^l with the sink vertex t_i. This constitutes a path between s_i and t_i, traversing all element edges of A_i. Note that all edges on this path that are not element edges can only be used by vertex-pair (demand) (s_i, t_i). Assign a capacity of 1 to all edges. Let $\boldsymbol{x}^T = (\underbrace{0, \dots, 0}_{n-K}, \underbrace{1, \dots, 1}_{K})$. This constitutes an instance of FIXMMF-MF, with n demands. Suppose that we have a positive answer to SET PACKING. This implies that there exist at least K mutually disjoint sets A_i. Assigning a flow of 1 to each of the corresponding vertex-pairs (s_i, t_i), and 0 to the rest will give $\vec{\boldsymbol{x}} = (\underbrace{0, \dots, 0}_{n-H}, \underbrace{1, \dots, 1}_{H})$, $H \geq K$. Thus $\vec{\boldsymbol{x}} \succeq \vec{\boldsymbol{x}}^T$, and we have a positive answer to FIXMMF-MF. Contrarily, suppose that we have a positive answer to FIXMMF-MF, i.e., that there exists a feasible allocation $\boldsymbol{x}$, with $\vec{\boldsymbol{x}} \succeq (\underbrace{0, \dots, 0}_{n-K}, \underbrace{1, \dots, 1}_{K})$. Since capacities are equal to 1, no edges can be shared by demands and $x_d \leq 1$ for all $d = 1, 2, \dots, D$. This implies that paths of demands that are assigned flow 1 must be disjoint. As constructed, these paths define at least K mutually disjoint sets A_i, and a positive answer to SET PACKING follows. Hence, since SET PACKING is $\mathcal{NP}$-complete, FIXMMF-MF is $\mathcal{NP}$-complete. □

Knowledge of the lifting algorithm and a bit of reflection reveals that the above result will not hold (unless $\mathcal{P} = \mathcal{NP}$) if $\boldsymbol{x}^T = c \cdot \boldsymbol{e}$, where $\boldsymbol{e}$ is the unity vector of size D, and c is a positive integer (corresponding to the optimization problem of finding the maximal first entry of the sorted allocation vector) nor if each demand's path has at most one shared link, both cases for which modified versions of the lifting algorithm solves the problems in polynomial time [8].

4 An algorithm

In this section we present an algorithm for the considered problem. The approach is in essence exploiting the "distribution approach" described for non-convex MMF problems in general in [9]. The distribution approach makes use of that if an MMF problem only has a discrete (finite) set of possible outcome values, then it can be stated as a lexicographic minimization. Even though the presented algorithm does not offer any general optimality or running time guarantees, we show that if the algorithm terminates with an integral solution, it must be optimal for the considered problem. In the following section this will be shown to happen quite frequently. If this is not the case, it is possible to modify the algorithm (although with the risk of making it heavily computationally complex) such that the output is guaranteed to be an optimal integral solution. The considered algorithm consists in successive resolution of Linear Programs (LPs). As there exist efficient methods for solving LPs (it will become evident that the algorithm solves at most r LPs with at most $D + rD$ variables and at most $rD + (r-1) + E$ constraints, where $r \leq \max_e\{c_e\}$) the algorithm may be an appealing way of approaching an instance of the problem. Define the linear programming problem P_k,

$$P_k: \qquad \tau_k = \min \sum_d t_{kd} \tag{2}$$

$$\text{s.t.} \quad l - x_d \le t_{ld}, \; l = 1,2,\ldots,k, \; d = 1,2,\ldots,D \tag{3}$$

$$\sum_d t_{ld} \le \lceil \tau_l \rceil, \; l = 1,2,\ldots,k-1 \tag{4}$$

$$\sum_d \delta_{ed} x_d \le c_e, \; e = 1,2,\ldots,E \tag{5}$$

$$x_d, t_{ld} \ge 0, \tag{6}$$

and consider Algorithm 1.

Algorithm 1

```
$k := 1$, improvement:=TRUE, $r := \max_e\{c_e\}$
**while** improvement and $k \le r$ **do**
  solve problem $P_k$ for $\boldsymbol{x}^*$ and $\tau_k$
  **if** $\max\{\boldsymbol{x}^*\} < k$ **then**
    improvement:=FALSE
  **end if**
  $k := k + 1$
**end while**
$\boldsymbol{x}' := \boldsymbol{x}^*$
```

Proposition 2. *Let $\boldsymbol{x}'$ be the output of Algorithm 1. If $\boldsymbol{x}' \in (\mathbb{Z}^+)^D$, then $\boldsymbol{x}' \in Q$, and $\boldsymbol{x}'$ is an optimal solution to the considered problem, i.e., $\vec{\boldsymbol{x}}' =$ lex max $\{\vec{\boldsymbol{x}} : \boldsymbol{x} \in Q\}$, where Q is the constraint set of (i) and (ii).*

Proof. Note that we may assign $r = \max_e\{c_e\}$, as for sure it must hold that $x_d \le \max_e\{c_e\}$ for all d, $d = 1,2,\ldots,D$. Note further that Proposition 2 does *not* require that solutions to intermediate LPs ($P_k, \; k < r$) are integral.

We will need some additional notation. As earlier we will use $\boldsymbol{y}$ to denote a sorted (in non-decreasing order) version of allocation vector $\boldsymbol{x}$, i.e, $\boldsymbol{y} = \vec{\boldsymbol{x}}$. Let $f : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ be the function for which $f(l,x) = \begin{cases} l - x & \text{if } \; l \ge x \\ 0 & \text{if } \; l < x \end{cases}$. Let $\boldsymbol{x}^i = (x_1^i, x_2^i, \ldots, x_D^i)$, $1 \le i \le r$, be a solution to

$$\text{lex max } \vec{\boldsymbol{x}} \tag{7}$$

$$\text{s.t.} \quad \sum_d \delta_{ed} x_d \le c_e, \; e = 1,2,\ldots,E \tag{8}$$

$$x_d \in \{0,1,\ldots,i\}, \; d = 1,2,\ldots,D, \tag{9}$$

and note that it always holds that $\sum_d f(i, x_d^i) = \sum_d f(i, x_d^j)$, for any (i,j) such that $1 \le i < j \le r$. For the proof we will assume that that the algorithm terminates after r iterations, since if it halts after u ($u < r$) iterations, we may redefine r as $r := u$. As the solution in that case does not improve for $k = u+1$, it follows that the (sorted) allocation vector being the solution to problem P_{u+j} cannot be lexicogaphically greater than the sorted solution to problem P_u, for any integer $j \geq 1$. Without loss of generality we may also assume that for a solution $\boldsymbol{x}^*$ to P_k it holds that $x_d^* \leq k$, for all $d = 1, 2, \ldots, D$, since if $\exists d : x_d^* > k$, we may assign $x_d^* = k$ for all such d, maintaining feasibility and objective function value. Further it must hold for $\boldsymbol{x}'$ that $\sum_d f(l, x'_d) = \lceil \tau_l \rceil$, for all l, $1 \leq l \leq r$. For $l = r$ this follows directly from the fact that $\tau_r = \lceil \tau_r \rceil$ (as $\boldsymbol{x}' \in (\mathbb{Z}^+)^D$) and that $\tau_r = \sum_d f(r, x'_d)$ by definition. For $l < r$, suppose that $\sum_d f(l, x'_d) > \lceil \tau_l \rceil$. Then $\boldsymbol{x}'$ is infeasible for P_r, which is a contradiction. On the other hand, suppose that i, $i < r$, is the smallest positive integer for which $\sum_d f(i, x'_d) < \lceil \tau_i \rceil$, holds. As $\boldsymbol{x}' \in (\mathbb{Z}^+)^D$, $\sum_d f(i, x'_d) \in \mathbb{Z}^+$ and $\sum_d f(i, x'_d) < \tau_i$ must be true, contradicting that τ_i is the optimal objective function value for P_i.

Let $\boldsymbol{x}^*$ be an optimal solution to P_1. We have that $\tau_1 = \sum_d f(1, x_d^*)$ and $\lceil \tau_1 \rceil = \sum_d f(1, x'_d)$, so $0 \leq \sum_d f(1, x'_d) - \sum_d f(1, x_d^*) < 1$. Now suppose that $\boldsymbol{y}^1 \succ \boldsymbol{y}'$. Then $\sum_d f(1, x'_d) - \sum_d f(1, x_d^1) \geq 1$ and thus $\sum_d f(1, x_d^1) < \sum_d f(1, x_d^*)$, which is a contradiction. Hence $\boldsymbol{y}' \succeq \boldsymbol{y}^1$.

Assume that $\boldsymbol{y}' \succeq \boldsymbol{y}^{k-1}$. Then $\boldsymbol{y}' \succeq \boldsymbol{y}^i$, $1 \leq i \leq k-1$. Let m be the number of entries of $\boldsymbol{x}^{k-1}$ that are strictly smaller than $k-1$. We will start by proving that $y_j^k = y'_j$ for $j \leq m$. Again aiming for contradiction, suppose that entry p, $p \leq m$, is the first entry for which $y_p^k \neq y'_p$. Then either $y_p^k < y'_p$ or $y_p^k > y'_p$. Suppose $y_p^k < y'_p = t$. This facilitates derivation of a solution $\boldsymbol{x}''$ from $\boldsymbol{x}'$ by assigning $x''_d = x'_d$ if $x'_d \leq t$ and $x''_d = t$ if $x'_d > t$. It will hold that $\sum_d \delta_{ed} x''_d \leq c_e$, $e = 1, 2, \ldots, E$, and that $x''_d \in \{0, 1, \ldots, t\}$, $d = 1, 2, \ldots, D$, and, which is contradictive, that $\boldsymbol{y}'' \succ \boldsymbol{y}^k \succeq \boldsymbol{y}^t$. On the other hand suppose that $y'_p < y_p^k = t$. Then $\sum_d f(t, x'_d) > \sum_d f(t, x_d^k)$. However, we also have that $\boldsymbol{y}' \succeq \boldsymbol{y}^t$ (as $t < k-1$), which implies that $\boldsymbol{y}' = \boldsymbol{y}^t$ or $\boldsymbol{y}' \succ \boldsymbol{y}^t$. If $\boldsymbol{y}' = \boldsymbol{y}^t$ then clearly $\sum_d f(t, x'_d) = \sum_d f(t, x_d^t) = \sum_d (t, x_d^k)$, which is a contradiction. If $\boldsymbol{y}' \succ \boldsymbol{y}^t$ then there must exist an entry s, such that $y'_s > y_s^t$ and $y'_i = y_i^t$, if $i < s$. Now $y_i^t = y_i^{k-1}$ for $i = 1, 2, \ldots, e$, so necessarily $s > p$. But this yields that $\sum_d f(t, x'_d) = \sum_d f(t, x_d^t) = \sum_d f(t, x_d^k)$, which is a contradiction. Hence, if $\boldsymbol{y}^k \succ \boldsymbol{y}'$ there must exist an entry q such that $y_q^k = k$, and $y'_q = k-1$. This in turn implies that $\sum_d f(k, x_d^k) < \sum_d f(k, x'_d) = \lceil \tau_k \rceil$. But $\sum_d f(i, x_d^k) = \sum_d f(i, x'_d) = \lceil \tau_i \rceil$, $1 \leq i \leq k-1$, as $y_i^k = y'_i$ for $i \leq m$. Hence τ_k cannot be the optimal value of the objective function for P_k, and summarizing, it cannot hold that $\boldsymbol{y}^k \succ \boldsymbol{y}'$. Thus $\boldsymbol{y}' \succeq \boldsymbol{y}^k$. By induction over k, it follows that $\boldsymbol{y}' = \boldsymbol{y}^r$. □

It is an immediate consequence that if we put as an explicit constraint in P_k, $k = 1, 2, \ldots, r$, that $x_d \in \mathbb{Z}^+$, $d = 1, 2, \ldots, D$, then the algorithm is guaranteed to solve problem (7)-(9) with $i = r$. Thus this is an option for an instance for which Algorithm 1 does not produce an integral solution. However, this makes P_k a Mixed Integer Programming (MIP) problem.

There are some implementational issues of Algorithm 1 that ought to be mentioned. First of all, it is convenient to recycle the sparse constraint matrices of the successive LPs, as they change only marginally between consecutive steps.

Secondly, special care should be taken in the rounding of τ. For large instances (many variables), aggregation of small numerical errors in the computed variables may cause an erroneous rounding of τ (typically making the right-hand side of (4) too large). Finally, it is essential that the LPs are solved for vertex solutions, as is done by Simplex. This is easily realized if one considers a network of two vertices connected by one edge of capacity 1. Assume that there are two demands between the vertices. As opposed to Simplex, an interior point solution to this instance cannot belong to $\{0,1\}^2$.

5 A numerical experiment

In this section we apply Algorithm 1 (the original LP-based version) to randomly generated problem instances ranging from 36 to 435 demands (corresponding to demands between all vertex-pairs in a 9-vertex network to all vertex-pairs in a 30-vertex network). In all instances $r = 50$ and each edge capacity, c_e, $e = 1, 2, \ldots, E$ belongs to $\{5, 10, 15, \ldots, 50\}$. The results can be found in Table 5. The first 4 columns give, in turn, the number of vertices, V, the number of edges, E, the number of demands, D, and the average length (hops) of a path, $\mathbb{E}(|p|)$. The 5:th column indicates if the algorithm halts with an integral solution (int). Column 6 gives the number of solved LP:s (iterations) that did not produce an integral solution (NILP), and the 7:th column gives the number of iterations for which τ was rounded up ($\lceil\tau\rceil$), i.e., when optimal τ was non-integral (due to rounding errors there is no one-to-one correspondence between rounded τ's and non-integral solutions). The following two columns give the total running time (t) and the required number of iterations (it), respectively. The columns are then repeated for more instances. The computations were carried out on a PC with an Intel PIII-1GHz CPU, RAM of 256 MB, and Windows 2000 OS. The algorithm was implemented in MATLAB6.5, and the LPs are solved using a MATLAB interface (mex-function) to CPLEX 9 (Simplex LP-solver).

V	E	D	$\mathbb{E}(\lvert p\rvert)$	int	NILP	$\lceil\tau\rceil$	$t(s)$	it	V	E	D	$\mathbb{E}(\lvert p\rvert)$	int	NILP	$\lceil\tau\rceil$	$t(s)$	it
9	20	36	5.3	yes	0	0	2.50	26	20	41	190	10.8	yes	0	0	14.95	28
10	20	45	4.7	yes	0	0	7.03	43	21	50	210	11.8	yes	1	1	47.22	42
11	21	55	6.4	yes	1	1	9.47	46	22	55	231	11.7	yes	0	1	40.28	36
12	27	66	6.8	yes	4	1	11.56	41	23	59	253	13.6	yes	1	14	53.77	40
13	28	78	7.2	yes	0	7	14.16	48	24	54	279	13.7	yes	2	10	55.41	37
14	34	91	7.7	yes	1	1	10.27	35	25	54	300	13.7	yes	0	0	43.43	32
15	29	105	9.0	yes	0	0	11.29	37	26	72	325	14.4	yes	0	0	147.34	50
16	34	120	8.6	yes	2	2	17.42	38	27	66	351	14.6	yes	0	0	38.77	26
17	39	136	8.8	yes	0	0	16.86	34	28	69	378	15.7	yes	0	0	144.45	42
18	43	153	10.3	yes	2	7	16.59	30	29	59	406	7.3	yes	0	0	351.72	49
19	45	171	10.3	yes	0	0	18.74	36	30	65	435	8.5	yes	1	0	426.30	49

Table 1. Testing the algorithm.

Although Algorithm 1 performs satisfactorily on all of the instances considered in Table 5, there exist situations for which it fails. Consider for instance the network given in Figure 3(c). Suppose that the same demands and paths as in Example 4 are given, and that edge capacities are all equal to 1. Then the (sorted) solution generated by Algorithm 1 is $\boldsymbol{y}' = (0.5, 0.5, 0.5)$ but the true OIS is $\boldsymbol{y}^z = (0, 0, 1)$.

6 Conclusions

This paper presents a study that addresses a realistic modification of the classical max-min fairness bandwidth assignment problem. As in the classical problem, we consider a set of node-pairs (demands) that are to be assigned bandwidth on fixed single paths in a capacitated network, such that the resulting distribution of flows among node-pairs is max-min fair. However, in this paper it is additionaly assumed that a demand can only be assigned a modular flow volume. The solution to the modified problem is first characterized. Then it is shown that even though the classical problem is solvable in polynomial time, the modified problem is $\mathcal{NP}$-hard. An algorithm based on linear programming (necessarily Simplex) is described and shown to be useful, both in terms of solution quality and running times, for a number of example instances. We prove that if this algorithm (in its basic linear programming form) produces an integral solution, then this must be the solution to the considered problem. It follows that the algorithm can, of course at the cost of increasing comlexity, instead be based on mixed integer programming, and then guarantee that the output is optimal.

References

1. Bertsekas, D., Gallager, R.: Data Networks. Prentice Hall (1987)
2. Nace, D.: A linear programming based approach for computing optimal fair splittable routing. In: IEEE International Symposium on Computers and Communications. (2002) 468–474
3. Pióro, M., Nilsson, P., Kubilinskas, E., Fodor, G.: On efficient max-min fair routing algorithms. In: IEEE International Symposium on Computers and Communications. (2003) 465–472
4. Pióro, M., Medhi, D.: Routing, Flow, and Capacity Design in Communication and Computer Networks. Morgan Kaufmann (Elsevier) (2004)
5. Kleinberg, J., Rabani, Y., Tardos, E.: Fairness in routing and load balancing. Journal of Computer and System Sciences **63**(1) (2001) 2–20
6. Nilsson, P., Pióro, M.: Unsplittable max-min demand allocation - a routing problem. In: HETNETs'05. (2005) P26.
7. Garey, M., Johnson, D.: Computers and intractability – a guide to the theory of NP-completeness. Freeman (1979)
8. Nilsson, P.: Some simple special cases of FIXMMF-MF. Technical report, Dept. of Communication Sytems, Lund University (2005)
9. Ogryczak, W., Pioro, M., Tomaszewski, A.: Telecommunications network design and max-min optimization problem. Journal of telecommunications and information technology **3** (2005)