Yugoslav Journal of Operations Research
29 (2019), Number 1, 93–112
DOI: https://doi.org/10.2298/YJOR181015034B

ON THREE APPROACHES TO LENGTH-BOUNDED MAXIMUM MULTICOMMODITY FLOW WITH UNIT EDGE-LENGTHS

Pavel BORISOVSKY
Sobolev Institute of Mathematics SB RAS, 13 Pevtsov str., 644099, Omsk, Russia
borisovski@mail.ru

Anton EREMEEV
Sobolev Institute of Mathematics SB RAS, 13 Pevtsov str., 644099, Omsk, Russia, and Dostoevsky Omsk State University, 55a pr. Mira, 644077, Omsk, Russia
eremeev@ofim.oscsbras.ru

Sergei HRUSHEV
Sobolev Institute of Mathematics SB RAS, 13 Pevtsov str., 644099, Omsk, Russia
hrushev@gmail.com

Vadim TEPLYAKOV
Yaliny Research and Development Center, 2 Paveletskaya naberezhnaya, block 2, 115114, Moscow, Russia
vadim.teplyakov@yaliny.com

Mikhail VOROZHTSOV
Yaliny Research and Development Center, 2 Paveletskaya naberezhnaya, block 2, 115114, Moscow, Russia
mikhail.vorozhtsov@yaliny.com

Received: August 2018 / Accepted: December 2018

Abstract: The paper presents a comparison between three approaches to solving the length-bounded maximum multicommodity flow problem with unit edge-lengths. Following the first approach, Garg and Könemann's, we developed an improved fully polynomial-time approximation scheme for this problem. As the second alternative, we considered the well-known greedy approach. The third approach is the one that yields exact solutions by means of a standard LP solver applied to an LP model on the time-expanded network.

Computational experiments are carried out on benchmark graphs and the graphs that model software defined satellite networks, to compare the proposed algorithms with an exact linear programming solver. The results of the experiments demonstrate a trade-off between the computing time and the precision of algorithms under consideration.

Keywords: Computational Experiment, Linear Programming, Fully Polynomial-Time Approximation Scheme, Greedy Heuristic, Software Defined Satellite Network.

MSC: 65K05, 68M10.

1. INTRODUCTION

Research issues in telecommunications often include requirements regarding quality of service such as bandwidth, delay, number of hops, etc. [24, 29]. These issues may be interpreted as combinatorial optimization problems of finding a multicommodity flow through the network that satisfies some quality of service constraints.

Multicommodity flow (MCF) problems are defined on a directed graph $G = (V, E)$ with edge capacities $u : E \to \mathbb{R}$ and k origin-to-destination pairs (s_j, t_j), $j = 1, \ldots, k$. These problems ask for a family of flows f_j from s_j to t_j, so that some optimization criterion is maximized under the node flow conservation constraints and the requirement that the sum of flows on any edge does not exceed the capacity of the edge.

In particular, the *maximum multicommodity flow* problem (*maximum MCF* for short) is an MCF problem where the total flow needs to be maximized. Some authors assume that the maximum MCF also includes a constraint that the value of each flow f_j is limited by a finite demand d_j, $j = 1, \ldots, k$ (see e.g. [29]).

A more general problem, called *length-bounded maximum multicommodity flow* (*length-bounded maximum MCF* for short), asks for a maximum MCF routed along a set of paths whose length does not exceed a specific bound. This problem was studied in [3, 5] assuming unbounded demands and in [9], assuming that finite demands are given. In particular, it was shown in [3] that the length-bounded maximum MCF is NP-hard, while its special case where all edges have unit length is polynomially solvable. An LP formulation for this special case was proposed in [21] using a multicommodity flow in a supplementary time-expanded network. In what follows, we will denote length-bounded maximum MCF problem by LBMCF and its special case where all edges have unit length will be denoted by LBMCF1. In this paper, we consider only one of the two well-known forms of the multicommodity max-flow problem, the so called *max-sum* one. The other tightly related form of this problem, called the *max-concurrent* version, with the objective to satisfy the maximum possible proportion of all demands, may be found e.g. in [4, 11, 13].

When the large-scale instances need to be solved, in view of excessive computational cost, it is often more appropriate to search for approximate solutions. A feasible solution y to a maximization problem is called a $(1 - \omega)$-*approximate* if it satisfies the inequality $f(y) \geq (1 - \omega)f^*$, where $0 < \omega < 1$ and f^* is the optimal

objective function value. An algorithm is called a $(1-\omega)$-*approximation algorithm* if in a polynomially bounded time, it outputs a $(1-\omega)$-approximate solution, given a solvable problem instance. A family of $(1-\omega)$-approximation algorithms parameterized by $\omega > 0$, such that the time complexity of these algorithms is polynomially bounded in $1/\omega$ and in the problem instance length is called *a fully polynomial-time approximation scheme (FPTAS)*.

A number of FPTASes have been developed for the maximum MCF problem, see e.g. [11, 13, 30]. FPTASes for the length-bounded maximum MCF were proposed in [3, 9]. In [29], an FPTAS was shown to exist for a more general *quality of service-aware MCF problem* (QoS-aware max-flow).

A modification of the length-bounded maximum MCF with an additional constraint that the flow on all edges must be integer-valued is called *integral* length-bounded maximum MCF. The results from [15] show that even when there is no length constraint at all, the edge capacities are equal to 1, and the graph is outerplanar, this problem does not admit approximation algorithms with constant approximation ratios, unless P=NP. A result from [16] implies that the integral length-bounded maximum MCF can not be approximated with a performance ratio $n^{0.5-\varepsilon}$ (where n is the number of nodes) for any $\varepsilon > 0$ even in the special case where all edges have equal length and all edge capacities are equal to 1, assuming P≠NP. In [18], a greedy $O(m^{0.5})$-approximation algorithm was proposed (where m is the number of arcs) for a special case of this problem, the unweighted edge-disjoint path problem. Complexity and non-approximability of the integral length-bounded maximum MCF was also studied in [3, 6].

The authors are not aware of preceding experimental studies devoted to LBMCF problems and LBMCF1 in particular. The exact and approximate approaches mentioned above were carefully studied in experiments with maximum MCF problems where the flows may be routed along the paths of unbounded length. The case of fractional flows was considered e.g. in [1, 4, 14, 25]. The integral maximum MCF and its version with unsplittable flows were studied experimentally in [2, 20, 22] and in other papers.

The present paper is aimed at an experimental comparison of between three approaches to approximate and exact solution of LBMCF1: the approach of Garg and Könemann to building FPTAS [13], the well-known greedy approach [2, 18, 19] and the exact solving by means of an LP model on a time-expanded network [21].

In Section 2, we give a statement of length-bounded maximum MCF and describe some of its basic properties. In Section 3, using the approach from [13] and its improvement from [11], we develop an FPTAS for a special case where all edges have unit length. This FPTAS has a smaller time complexity compared to the FPTAS from [29] which was developed for arbitrary edge lengths. In Section 4, we propose a simple greedy heuristic applicable to length-bounded maximum MCF as well as integral length-bounded maximum MCF (assuming unit edge lengths in both cases). In Section 5, we compare these algorithms with each other and with CPLEX solver in computational experiments. The last two sections contain the further research directions and conclusions. Appendix contains some proofs omitted in the text.

2. PROBLEM FORMULATIONS AND BASIC PROPERTIES

A flow in a digraph $G = (V, E)$ from origin vertex $s \in V$ to destination vertex $t \in V$ is a nonnegative function $f : E \to R_+$ such that for each node $v \in V$, $v \neq s$, $v \neq t$ holds

$$\sum_{(v',v)\in E} f(v', v) - \sum_{(v,v')\in E} f(v, v') = 0$$

and

$$|f| = \sum_{(s,v')\in E} f(s, v') - \sum_{(v',s)\in E} f(v', s) = \sum_{(v',t)\in E} f(v', t) - \sum_{(t,v')\in E} f(t, v')$$

is the amount of flow sent from s to t in f. If $P_1, \ldots, P_r$ are paths from s to t, then a sum of path-flows along $P_1, \ldots, P_r$ gives a network flow from s to t again. Given that $f = \sum_{j=1}^{r} f^{P_j}$, we will say that f *is routed along the set of paths* $P_1, \ldots, P_r$.

We assume that flow f_i of commodity $i, i = 1, \ldots, k$, has an origin $s_i \in V$ and a destination $t_i \in V$. If $f_1, f_2, \ldots, f_k$ are flows of k commodities, then $F = (f_1, f_2, \ldots, f_k)$ is called a *multicommodity flow* in G.

An input instance of the maximum MCF consists of a directed network $G = (V, E)$, where $|V| = n$, $|E| = m$, an edge capacity function $u : E \to \mathbb{R}_+$ and a specification $(s_i, t_i, d_i) \in V \times V \times \mathbb{R}^+$ of commodity i for $i = 1, \ldots, k$. The objective is to maximize $\sum_{i=1}^{k} |f_i|$, so that the sum of flows on any edge $e \in E$ does not exceed $u(e)$ and $|f_i| \leq d_i$, $i = 1, \ldots, k$.

The length-bounded maximum MCF has the same input, extended by an upper bound $L \in \mathbb{Z}^+$ and the edge lengths $\tau(e)$, $e \in E$ and asks for a maximum MCF where the sum of flows on any $e \in E$ does not exceed $u(e)$, $|f_i| \leq d_i$, $i = 1, \ldots, k$, and the flow of each commodity is routed along a set of paths, where each path has a length at most L. A special case of this problem where all edges have a unit length requires that the flow of each commodity is routed along a set of paths at most L edges long. Obviously, the maximum MCF may be considered a special case of LBMCF1, assuming $L = n$. W.l.o.g. we will assume that all pairs (s_i, t_i) are unique, since otherwise the demands with identical pairs (s_i, t_i) may be summed together in one demand.

LBMCF1 is polynomially equivalent to its special case with unbounded demands. Indeed, given an LBMCF1 instance $G = (V, E)$, $u : E \to \mathbb{R}_+$, (s_i, t_i, d_i), $i = 1, \ldots, k$ and L, one can consider a new instance of this problem with unbounded demands on a network G', obtained from G as described below. Let $T_v \subset V$ be the set of all vertices where at least one commodity originating in v is consumed, i.e. $T_v := \cup_{i:s_i=v}\{t_i\}$. For each vertex $v \in V$ with $|T_v| > 0$:

- New $|T_v|$ vertices are created and connected by arcs leading into vertex v.
- The new vertices are assigned to commodities with destinations in T_v by a one-to-one mapping. So we can denote the new vertices connected to v as v_i, where i is such that $s_i = v$.
- The capacities of edges leading from vertices v_i to v are set to d_i.

In the new problem the demands are $(v_i, t_i, +\infty)$, $i = 1, \ldots, k$, and the new upper bound is $L' = L + 1$.

There is a bijection between the sets of feasible solutions of the original instance and the new instance, and the objective function values of the corresponding solutions are equal. Now, since the special case of LBMCF1 with unbounded demands is solvable by LP methods in weakly polynomial time [3], the above reduction implies that LBMCF1 with finite demands is also solvable in weakly polynomial time (the same follows from the LP formulation (5)–(9) given below). The question of solvability of LBMCF1 in strongly polynomial time remains open. In the special case of maximum MCF, a strongly polynomial algorithm is known [28]. Nevertheless, even in this special case, existence of exact algorithms using the same combinatorial techniques as the Ford-Fulkerson method for the single-commodity flow is unlikely (see e.g. [26], § 70.13).

In the special case of maximum MCF there is a well-known formulation of the problem in terms of the LP using edge flows (see e.g. [17], § 11) with $O(k + m + kn)$ constraints and $O(km)$ variables. Assuming that variables $x_i(e) \geq 0$ give the amount of flow of commodity i over edge e the LP model is as follows.

$$\max \sum_{i=1}^{k} \sum_{e=(s_i,v)\in E} x_i(e), \tag{1}$$

$$\sum_{e=(s_i,v)\in E} x_i(e) \leq d_i, \quad i = 1, \ldots, k, \tag{2}$$

$$\sum_{i=1}^{k} x_i(e) \leq u(e), \quad e \in E, \tag{3}$$

$$\sum_{e=(v',v)\in E} x_i(e) = \sum_{e=(v,v')\in E} x_i(e), \quad i = 1, \ldots, k, \quad v \in V \backslash \{s_i, t_i\}, \tag{4}$$

An LP formulation of LBMCF1, involving $O(Lkn + m)$ constraints and $O(Lkm)$ variables, may be constructed using a multicommodity flow in a supplementary time-expanded network [21]. The node set V' contains a copy V_t of the node set V of graph G for every discrete time step $t, t = 1, \ldots, L$. For every directed edge $(v, w) \in E$, there is an edge in E' from vertex $v_t \in V_t$ in time layer t to vertex $w_{t+1} \in V_{t+1}$. Besides, E' contains edges (v_t, v_{t+1}) for all $v_t \in V_t$, $t = 1, \ldots, L-1$. A multicommodity flow is sought in this time-expanded network under additional constraints which require that for each $e = (v, w) \in E$ the sum of all flows traversing the edges (v_t, w_{t+1}), $t = 1, \ldots, L-1$ is at most $u(e)$. For all $i = 1, \ldots, k$, the origin of commodity i is placed in the copy s_{i1} of vertex s_i at level 1 and the destination

is placed in the copy t_{iL} of vertex t_i at level L. The resulting LP formulation is as follows

$$\max \sum_{i=1}^{k} \sum_{e'=(s_{i1},v_2)\in E'} x_i(e'), \tag{5}$$

$$\sum_{e'=(s_{i1},v_2)\in E'} x_i(e') \le d_i, \quad i = 1,\dots,k, \tag{6}$$

$$\sum_{i=1}^{k} \sum_{e'=(v_t,w_{t+1})\in E'} x_i(e') \le u(e), \quad e = (v,w) \in E, \tag{7}$$

$$\sum_{e'=(v_{t-1},w_t)\in E'} x_i(e') = \sum_{e'=(w_t,v_{t+1})\in E'} x_i(e'), \tag{8}$$

$$i = 1,\dots,k, \quad w_t \in V_t, \quad t = 2,\dots,L-1,$$

$$\sum_{e'=(v_{L-1},w_L)\in E'} x_i(e') = 0, \quad i = 1,\dots,k, \quad w_L \in V_L\backslash\{t_{Li}\}, \tag{9}$$

where variables $x_i(e') \ge 0$ give the amount of flow of commodity i over edge $e' \in E'$.

The practice shows that large MCF problems require a long time and a great amount of memory to solve using the exact LP methods either in path flow-based or edge flow-based formulations. (see e.g. [26], § 70.13). For this reason, it is important to develop faster algorithms to solve MCF problems approximately and LBMCF1 among them.

3. FULLY POLYNOMIAL TIME APPROXIMATION SCHEME

3.1. The Case of Unbounded Demands

This subsection presents an FPTAS for LBMCF1 with unbounded demands, which is developed analogously to the FPTAS for maximum MCF with unbounded demands [11].

Let $\mathcal{P}_i$ denote the set of all paths from s_i to t_i in G and let $\mathcal{P}_i(L)$ denote the subset of $\mathcal{P}_i$ which consists of paths at most L edges long. Besides that, put $\mathcal{P}(L) = \cup_{i=1}^{k} \mathcal{P}_i(L)$. The main difference from the preceding algorithms [11, 13] is that instead of $\mathcal{P}_i$, here we use $\mathcal{P}_i(L)$, $i = 1,\dots,k$.

LBMCF1 in the case of unbounded demands may be formulated as an LP problem (denoted by **P**) with an exponential number of path flow variables $x(P)$, $P \in \mathcal{P}(L)$:

$$\max \sum_{P \in \mathcal{P}(L)} x(P), \tag{10}$$

$$\sum_{P \in \mathcal{P}(L): e \in P} x(P) \le u(e), \quad e \in E, \tag{11}$$

$$x(P) \ge 0, \quad P \in \mathcal{P}(L). \tag{12}$$

The dual problem with a polynomial number of variables $y(e) \ge 0, \quad e \in E$ is

$$\min \sum_{e \in E} u(e) y(e), \tag{13}$$

$$\sum_{e \in P} y(e) \ge 1, \quad P \in \mathcal{P}(L), \tag{14}$$

$$y(e) \ge 0, \quad e \in E, \tag{15}$$

We first describe the general ideas of the $(1 - \omega)$-approximation algorithm for problem **P** according to the framework of Garg and Könemann [13], and after that a faster version will be described in detail.

The algorithm proceeds by iterative improvements of primal solutions to problem **P** simultaneously computing a sequence of dual feasible solutions. The latter ones allow to estimate the precision of the current primal solution and to find directions for further improvement. It is convenient to compute a set of parameters $\{\ell(e)\}_{e \in E}$, called *edge lengths* instead of the current dual-feasible solution y. In what follows, $\ell(P)$ denotes the sum of lengths of all edges comprising a path P. The dual feasible y may be reconstructed by scaling $y(e) = \ell(e)/\alpha, \ e \in E$, where the factor α is the length of a shortest path in $\mathcal{P}(L)$.

The algorithm starts with length function $\ell(e) = \delta$ for all $e \in E$ using some $\delta > 0$, and with a primal solution $x(P) = 0, P \in \mathcal{P}(L)$. While there is a path in $\mathcal{P}(L)$ of length less than 1, the algorithm selects such a path and updates the primal and the dual variables as follows. For the primal solution x, the algorithm increases the flow along path P by the minimum edge capacity in the path, i.e. denoting this bottleneck capacity by u, we update the primal solution by setting $x(P) = x(P) + u$. The updated primal solution may be infeasible, so in order to return x to feasible region, all of its components are scaled down by an appropriate scalar. After that the dual solution is updated so that the higher the congestion of an edge, the greater multiplier is given to its length:

$$\ell(e) = \ell(e)\left(1 + \frac{\varepsilon u}{u(e)}\right) \ e \in P,$$

(the choice of the parameter $\epsilon \in (0, 1]$ will be discussed in what follows). With such an update rule, the length of the bottleneck edge always increases by a factor of $(1 + \varepsilon)$, while the lengths of edges not on P remain unchanged.

It will be shown in the sequel that this algorithm makes a polynomially bounded number of iterations, and each iteration increases $x(P)$ for just one path P, therefore the number of non-zero components in the computed solution x is polynomially bounded. Note that upon completion of this algorithm all paths in $\mathcal{P}(L)$ have a length at least 1, so the set of parameters $\{\ell(e)\}_{e\in E}$ is a dual feasible solution without any scaling.

In order to find out if there is a path $P \in \mathcal{P}(L)$ of length $\ell(P) < 1$ according to the current length function, it suffices to compute a shortest path in $\mathcal{P}(L)$ for each commodity. This may be done by executing L iterations of the Bellman-Ford algorithm in $O(kLm)$ time (see e.g. Theorem 2.3 in [23]). Let $\alpha := \min_{P\in\mathcal{P}(L)} \ell(P)$ denote the current shortest path length in $\mathcal{P}(L)$ and let $\hat{\alpha}$ be a lower bound on α, which will be evaluated implicitly at each iteration of the algorithm as described below.

Instead of looking through all origin-destination pairs of commodities, seeking for a shortest path in $\mathcal{P}(L)$, we implement the improvement of Fleischer [11], which consists in using a path of length at most $(1 + \varepsilon)\alpha$ and spending less time to find such a path. To this end, we cycle through all commodities, staying with one commodity until the shortest origin-to-destination path for that commodity in $\mathcal{P}(L)$ is above $(1 + \varepsilon)\hat{\alpha}$. The initial lower bound $\hat{\alpha}$ is set to δ. As long as there is some path $P \in \mathcal{P}(L)$ of length $\ell(P) < \min\{1, (1 + \varepsilon)\hat{\alpha}\}$, we augment the flow along such P. When such a path does not exist, it means that either $\alpha \geq 1$ and it is time to terminate the algorithm or $\alpha < 1$ and $\alpha \geq (1 + \varepsilon)\hat{\alpha}$. In the latter case, one can update the lower bound by setting $\hat{\alpha} := (1 + \varepsilon)\hat{\alpha}$. With such updates, the lower bound will belong to the set $\{\delta(1 + \varepsilon)^r\}_{r=0,1,2,\ldots}$. Upon the termination of the algorithm, $\hat{\alpha} \in [1, 1 + \varepsilon]$. Since each time $\hat{\alpha}$ is increased by a factor of $(1 + \varepsilon)$, the number of times that this happens is $\lfloor \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} \rfloor$, where $\lfloor\cdot\rfloor$ denotes the rounding down. This implies that the final value of r is $r_{\max} = \lfloor \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} \rfloor$. Postulating that every new value of the lower bound $\hat{\alpha}$ defines a new phase r of the algorithm, we will have an FPTAS represented by Algorithm 3.1 with the main loop over phases $1, \ldots, r_{\max}$ as described below. A detailed outline of the algorithm follows the $(1 - \omega)$-approximation algorithm for maximum MCF [11] with minor modifications.

In what follows we assume that S is the set of all vertices where at least one commodity originates, i.e. $S := \cup_{i=1}^{k} \{s_i\}$. If there is a vertex $v' \in T_v$ such that no path from $\mathcal{P}_i(L)$ leads from v to v', then the corresponding demand can not be served at all. All such pairs of vertices v, v' may be identified at the preprocessing stage and the corresponding flows should be set to zero. W.l.o.g. we will assume that such pairs of vertices do not exist.

Algorithm 3.1. $(1-\omega)$**-approximation algorithm for LBMCF1 with unbounded demands**

Initialization
Choose ε, δ and assign $r_{max} := \lfloor \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta} \rfloor$. Assign $l(e) := \delta$ for all $e \in E$.

The main loop
For all $r = 1, ..., r_{max}$ **do**
For all $v \in S$ **do**
Find $|T_v|$ shortest paths $P(v') \in \mathcal{P}(L)$ from v to all $v' \in T_v$.
Choose $v^* = \arg\min_{v' \in T_v} \ell(P(v'))$. Let $P = P(v^*)$.
While $\ell(P) < \min\{1, \delta(1+\varepsilon)^r\}$ **do**
Let $u := \min_{e \in P} u(e)$.
For all $e \in P$ assign $\ell(e) := \ell(e)\left(1 + \frac{\varepsilon u}{u(e)}\right)$.
Augment the path-flow $x(P) := x(P) + u / \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$.
Find $|T_v|$ shortest paths $P(v') \in \mathcal{P}(L)$ from v to all $v' \in T_v$.
Choose $v^* = \arg\min_{v' \in T_v} \ell(P(v'))$. Let $P = P(v^*)$.
End while
End for-loop over v**.**
End for-loop over r.

In the above algorithm, the length-bounded shortest paths from v are found by performing only L iterations of Bellman-Ford algorithm. The value of the flow computed by Algorithm 3.1 and its number of iterations are bounded analogously to those of $(1-\omega)$-approximation algorithm for the maximum MCF [11]. For this reason the proof of Theorem 3.4 (see below), and the proofs of its supplementary lemmas, are moved to the Appendix.

Lemma 3.2. *Algorithm 3.1 terminates after* $O\left(m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}\right)$ *augmentations.*

Lemma 3.3. *The flow* $f(e) = \sum_{P \in \mathcal{P}(L): e \in P} x(P)$, $e \in E$ *obtained by Algorithm 3.1 satisfies the constraints* $f(e) \le u(e)$ *for all* $e \in E$.

Theorem 3.4. *(i) Algorithm 3.1 makes* $O\left(m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}\right)$ *augmentations.*

(ii) Given $\delta = \frac{1+\varepsilon}{\sqrt[\varepsilon]{(1+\varepsilon)L}}$, *the feasible solution to LBMCF1 with unbounded demands found by Algorithm 3.1 has a total flow value* $g \ge \frac{f^*(1-\varepsilon)^2}{1+\varepsilon}$.

Now suppose a value $\omega > 0$ is given. Then choosing $\varepsilon = \frac{3-\omega-\sqrt{(3-\omega)^2-4\omega}}{2}$ and $\delta = \frac{1+\varepsilon}{\sqrt[\varepsilon]{(1+\varepsilon)L}}$, by Theorem 3.4, part (ii) we conclude that the obtained flow has a value at least $(1-\omega)f^*$.

The length-bounded shortest paths from vertex v are computed once in the beginning of each iteration in the loop over $v \in S$, and it is computed after

each augmentation. There are at most $|S| \cdot r_{max} = O(\varepsilon^{-2} n \log(L))$ applications of the truncated Bellman-Ford algorithm in Algorithm 3.1 that do not lead to augmentations. Besides, Theorem 3.4, part (i) implies that the truncated Bellman-Ford algorithm is applied at most $O(m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}) = O(\varepsilon^{-2} m \log(L))$ times with augmentations. Thus the total runtime of Algorithm 3.1 is $O(\varepsilon^{-2} m^2 L \log(L))$ and we have the following

Corollary 3.5. *Given $\omega \in (0,1)$, Algorithm 3.1 with $\varepsilon = \frac{3-\omega-\sqrt{(3-\omega)^2-4\omega}}{2}$ and $\delta = \frac{1+\varepsilon}{\sqrt[\varepsilon]{(1+\varepsilon)L}}$ computes a $(1-\omega)$-approximate solution to LBMCF1 with unbounded demands in $O(\omega^{-2} m^2 L \log L)$ time.*

Regarding the algorithm described in this subsection, we note that the idea of adapting the approach from [11] to the length-bounded MCF has already been discussed by Baier [3] for the case with general edge lengths. It was already noted in [3] that the resource-constrained shortest path problem, which arises as a subproblem in the general case, is NP-hard and can only be solved approximately. In our algorithm, we exploit the fact that in the unit edge length case, the subproblem becomes efficiently solvable.

3.2. The Case of Finite Demands

The reduction described in Section 2 allows to obtain $(1-\omega)$-approximate solutions to LBMCF1 by applying Algorithm 3.1 to the transformed instance which consists of network G' with $n' = n + k$ vertices and $m' = m + k$ edges, the specification of commodities $(v_i, t_i, +\infty)$, $i = 1, \ldots, k$ and the length bound $L' = L + 1$. We can save some time by using the structure of G'. Here one can find the shortest paths consisting of at most L' edges from all the new vertices v_i attached to a vertex v, $v \in S$, by means of L iterations of Bellman-Ford algorithm, starting it from v. Then in total there are at most $O(\varepsilon^{-2} n \log(L))$ calls of the Bellman-Ford algorithm that do not lead to an augmentation and there are at most $O(\varepsilon^{-2}(m + k) \log(L))$ calls to the Bellman-Ford algorithm following the augmentations, so we have

Corollary 3.6. *A $(1-\omega)$-approximate solution to LBMCF1 may be computed in time $O(\omega^{-2}(m+k)mL \log L)$.*

This corollary gives a lower time bound compared to the time bound $O(\omega^{-2}(m+k)mn \log(m+k) \cdot (\log\log n + 1/\omega))$ of the FPTAS developed in [29]. However the FPTAS from [29] is applicable to LBMCF where the length function has a more general form.

4. GREEDY HEURISTIC

As an alternative to the guaranteed approximation algorithm from Subsection 3.2, we consider a simple Greedy heuristic, based on augmenting flows along

the shortest paths. This is a well-known greedy approach, which proved to be fruitful for approximate solution of different versions of maximum MCF with an additional constraint that the flow on all edges must be integer-valued or each commodity should be routed along a single path [2, 16, 18, 19]. In general, the idea of greedy augmenting a flow via shortest path flows dates back to the $O(n^2m)$-time algorithm for the maximum flow problem [10].

In what follows, by a shortest path we mean a path with the minimal number of edges in G. In each iteration of Greedy, the shortest paths are found for all origin-destination pairs and a maximum possible flow is routed along the path of minimal length among the paths with at most L edges. After that we decrease the edge capacities along the path by the value of the path flow, delete all edges where the remaining capacity turns to 0 and proceed to the next iteration. The algorithm terminates when a set $\mathcal{S}$ of shortest paths from $\mathcal{P}(L)$, connecting the unsatisfied origin-destination pairs in the current network, becomes empty. Here we assume that $\mathcal{P}(L)$ is the set of paths at most L edges long *in the current network.*

Algorithm 4.1. Greedy Heuristic for LBMCF1

Initialization of set $\mathcal{S}$:
Compute a shortest path $P_{s_it_i} \in \mathcal{P}(L)$ from s_i to t_i
for all $i = 1, \ldots, k$, for which such paths exist.
Denote the set of computed paths by $\mathcal{S}$.
The main loop:
While $|\mathcal{S}| > 0$ **do**
Choose a shortest path P in $\mathcal{S}$, put $\mathcal{S} := \mathcal{S} \backslash P$.
Let s_i and t_i be the first and the last vertices in path P.
$u_{\min} := \min\{u(e) : e \in P\}$, $x(P) := \min\{u_{\min}, d_i\}$.
$d_i := d_i - x(P)$.
For all $e \in P$ **do** $u(e) := u(e) - x(P)$.
If there are edges $e \in P$, such that $u(e) = 0$ **then**
Delete all edges $e \in P$, such that $u(e) = 0$, from G.
Build a new set $\mathcal{S}$ by computing shortest paths $P_{s_it_i} \in \mathcal{P}(L)$
from s_i to t_i for all $i = 1, \ldots, k$, for which such paths exist.
End if.
End while.

Clearly, the collection of path-flows with $x(P) > 0$ found by Greedy constitutes a feasible solution. Besides that, since the set $\mathcal{S}$ of length-bounded shortest paths is computed at most m times, the time complexity of Greedy heuristic is $O(m^2 \log n)$ if the Dijkstra algorithm with heaps is used. If the truncated Bellman-Ford algorithm is used, then the time complexity of Greedy is $O(m^2L)$.

Greedy can easily be converted to compute approximate solutions for length-bounded maximum MCF if instead of the number of edges in a path one takes the path length in terms of edge lengths $\tau(e)$. Note that Greedy outputs a feasible solution with an integer-valued flow on all edges if an instance is feasible and all demands and edge capacities are integer-valued.

5. COMPUTATIONAL EXPERIMENTS

This section describes the computational experiments in solving LBMCF1 by $(1-\omega)$-approximation algorithm from Subsection 3.2, by Greedy heuristics from Section 4, and by the LP-solver CPLEX 11 in dual simplex mode, using the LP formulation based on the time-expanded networks. For the large instances, where this formulation required too much time and memory, we used the LP formulation of Maximum MCF from [17] to find an upper bound for the optimum. All experiments were carried out on Xeon X5675, 3.07 GHz, 96 Gb RAM, 12 cores.

5.1. Implementation Details

The $(1-\omega)$-approximation algorithm for LBMCF1 was implemented with a minor improvement [11], which allows to terminate the algorithm when the best-found primal solution and the best-found dual feasible solution are within the required approximation ratio. Whenever the total amount of some commodity i, routed by the current iteration, achieves the corresponding demand d_i, this commodity is excluded from consideration in subsequent iterations.

It is easy to see that the most time-consuming part of the Greedy algorithm is the procedure building the sets of shortest paths. We found that this part of Greedy is so compute-intensive that CPU cache misses percent may slow down the execution significantly, so we reduced the memory footprint of the all-pairs shortest paths procedure by implementing compact data structures both for the input graph and for the temporary data. Also, we used a specially designed version of the Dijkstra's algorithm that may be called in parallel for each root vertex $v \in V$. With these optimizations, we reduced the final time-footprint of the Greedy up to a factor 0.1 of its first naïve implementation.

5.2. Problem Instances

The networks for testing instances were obtained using a modification of generator RMFGEN [14]; besides, several instances with real-life structure were constructed. RMFGEN produces a given number of two-dimensional grids with arcs connecting a random permutation of nodes in b adjacent planar grids of size $a \times a$. Arcs of the grids have capacities 3600, while the capacities of arcs connecting the grids are chosen uniformly at random from 1 to 100. Origins and destinations are randomly chosen. In all experiments with RMFGEN networks, the demands were assigned as in [14] so that there is a feasible flow with the maximum arc congestion λ, where λ is set to 0.6 or 1.

The instances with real-life structure represent prospective Software Defined Satellite Networks (SDSN) (see e.g. [27]), which provide world-wide telecommunication services. We suppose that the network consists of 135 satellites on low Earth orbit (LEO), 40 ground stations (gates to Internet), and a Network Operations Control Center (NOCC). In this SDSN, the packet routes for each origin-to-destination pair (s_i, t_i) will be computed at NOCC in real time and each node (satellite or ground station) will regularly receive the updated routes for all packets that originate in this node. Each packet sent from s_i to t_i contains some

content data and a path of the packet route from s_i to t_i. An upper bound L on the number of edges in packet paths is imposed due to a natural technical limitation on the number of bits reserved for encoding a packet route. Short packet paths are also preferable because they tend to have low transmission delays. For simplicity, we assume that each problem instance describes the system in a single time-frame and all demands for the time-frame are known in advance.

The original input data is not the same as for the considered LBMCF1 problem. The communications between satellites and between satellites and stations are represented as usual, as the directed arcs with given bandwidth capacities but the communications between stations are different. The station consists of several internet gateways that can be connected with the gateways of the other stations. The bandwidth is defined in total for the whole gateway rather than for the particular arcs between gateways.

The graphs modelling the SDSN were constructed with different trade-off between the model accuracy and the size of G. Networks of these instances contain a component with vertices of low degree (from 2 to 7) corresponding to satellites, and a component of vertices with high degree (near to $n/3$) corresponding to Internet gates at ground stations.

In Instance 1 of the series, each gateway is represented by its own "main" node, and in addition, has two artificial nodes for incoming and outgoing traffic. The gateway bandwidth is assigned to the arcs from the main node to these artificial nodes. The traffic between gateways is passed only through the artificial nodes. This is an exact representation, but has the largest size of the resulting graph G.

In Instance 2, each gateway is represented by one node and is connected with one artificial "central" node forming the star subgraph on the set of the gateway nodes. Instance 3 is the same as 2 but with doubled bandwidths. In Instance 4, all the gateways of the same station are united in one station node and are connected with all other such nodes, forming a clique on the set of station nodes. Instance 5 is the same as 4 but with doubled bandwidths. In Instance 6, all the stations are removed, only the satellites are given. In Instance 7 all the gateways of one station are united in one gateway having their total bandwidth, then the same approach as for Instance 1 is applied.

The demands for commodities were generated so as to model the global telecommunication flows. We assumed that the number of active users in each square unit of the Earth surface is proportional to population on the unit. The origin and the destination of each call are chosen at random among active users. All active users are assigned to the nearest satellite or ground station.

The input data of the test instances can be downloaded from:
http://math.nsc.ru/AP/benchmarks/Flow/mcf_tests.html

5.3. Experimental Results

The values of relative errors of solutions found by FPTAS and by Greedy heuristic were estimated a-posteriori in terms of upper bound $\omega' := (UB - f_{\text{appr}})/UB$, where f_{appr} is the value of objective function found by the algorithm and UB is the optimum in the LP model based on the time-expanded network (on grid

Instance	1	2	3	4	5	6	7
n	543	272	272	197	197	135	318
m	19474	1292	1292	992	992	750	4652
k	12373	12373	12373	12373	12373	743	12373

Table 1: Parameters of Instances With Real-Life Structure

graphs) or an upper bound on the optimum in maximum MCF LP formulation (on the instances with real-life structure).

5.3.1. Experiments on Grid Graphs

The attained approximations ω' and CPU times (in seconds) for the grid graphs with $a = 6$ and $k = 15$ are given in Table 2. One can see that the FPTAS occupies the position between Greedy heuristic and CPLEX solver both in terms of the precision and the running time, except for the two smallest instances where Greedy was able to find optimal solutions. The parallel version of Greedy using 12 cores is clearly the fastest one, achieving speed-ups of about 4.5 times compared to the serial version. On small instances this speed-up vanishes due to the communication cost.

The growth of CPU times with further increase of k is displayed in Fig 1. Here we use the largest grid graph with $a = 6$, $b = 8$. Both approximate algorithms have run-time upper bounds independent of k (see Sections 3 and 4), and we can see that the curves of actual run-times of these algorithms in Fig 1 are nearly horizontal. The size of LP formulation based on the time-expanded network depends significantly on k, which is supported by Fig 1.

RMFGEN instance			Greedy			FPTAS, $\omega = 0.2$		CPLEX
b	λ	$\|E\|$	CPU time		ω'	CPU time	ω'	CPU time
			1 core	12 cores		1 core		1 core
2	0.6	276	0.01	0.01	**0**	0.07	0.01	0.15
4	0.6	588	0.05	0.02	**0**	0.23	0.01	0.25
6	0.6	900	0.09	0.02	0.06	0.53	**0.02**	0.64
8	0.6	1212	0.14	0.03	0.29	0.74	**0.02**	1.06
2	1	276	0.01	0.01	**0.03**	0.08	0.04	0.15
4	1	588	0.06	0.02	0.06	0.3	**0.05**	0.61
6	1	900	0.12	0.04	0.07	0.57	**0.01**	0.44
8	1	1212	0.16	0.04	0.28	0.79	**0.02**	0.59

Table 2: A posteriori estimated approximation ω' and CPU times on grid graphs, $L = 9$

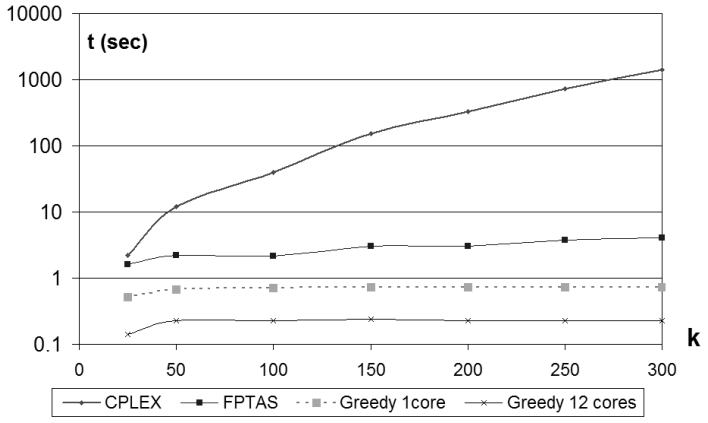

Figure 1: CPU time as a function of k on grid graph, $a = 6$, $b = 8$.

Inst-ance	Greedy heuristic			FPTAS, $\omega = 0.2$,		CPLEX computing *UB*
	CPU time 1 core	CPU time 12 cores	ω'	CPU time	ω'	CPU time
1	1.63	0.4	0.077	1689.6	**0.004**	–
2	0.51	0.16	0.054	157.6	**0.008**	7 971
3	0.23	0.1	0.057	160.4	**0.01**	6 947
4	0.11	0.05	0.038	80.6	**0.012**	1 852
5	0.07	0.04	0.026	83.6	**0.012**	1 872
6	0.18	0.1	0.177	3.8	**0.12**	345
7	0.23	0.06	0.039	389.5	**0.006**	144 456

Table 3: A posteriori estimated approximation ω' and CPU times on instances with real-life structure, $L = 9$

5.3.2. Experiments on Instances with Real-Life Structure

Instances 1-7 have a greater range of graph sizes and a much greater number of commodities compared to the instances with grid graphs (see Tables 1 and 2).

The LP model based on the time-expanded network required prohibitive amount of time and memory. Therefore in the case of Instances 1-7 we could compute only an upper bound *UB* using the LP solver of CPLEX. Still, in the case of Instance 1, CPLEX was unable to find an upper bound due to lack of memory and *UB* was set to the total value of demands.

Table 3 shows the a posteriori pessimistic estimates of approximations ω' attained by the algorithms and the corresponding CPU times (in seconds). Here FPTAS always has a greater precision than Greedy and the latter one is up to 10^3 times faster even in the sequential version.

In order to evaluate the algorithms on a variety of different real-life instances

	Greedy	FPTAS		
		$\omega = 0.4$	$\omega = 0.2$	$\omega = 0.1$
Average	0.245	164.8	402.0	857.6
Maximum	0.3	177.6	444.1	888.7

Table 4: CPU times (sec.) on 300 instances with real-life structure, $L = 9$

	Greedy	FPTAS		
		$\omega = 0.4$	$\omega = 0.2$	$\omega = 0.1$
Average	0.038	0.006	0.004	0.003
Maximum	0.076	0.01	0.006	0.004

Table 5: A posteriori estimated approximation ω' on 300 instances with real-life structure, $L = 9$

with similar structure we generated 300 versions of graph G that model SDSN analogously to Instance 7 but in different time-frames. Different satellite positions in these time-frames lead to different links between satellites and between satellites and ground stations. The bandwidth of the links varies as well. The specification of demands remained unchanged. Application of FPTAS and Greedy to these instances in the single-core version with different values of parameters ω and L gave the results shown in Tables 4, 5 and 6.

Tables 4 and 5 indicate that the average and maximum CPU times, as well as ω' estimates are close to those reported in Table 3 for Instance 7, which implies that both algorithms have a stable behavior on this type of input data.

One can see from Table 6 that increasing L increases the CPU time of FPTAS but reduces the CPU time of Greedy. Clearly, when L is low, the FPTAS time reduces due to fewer number of iterations of the truncated Bellman-Ford algorithm. On the other hand, in Greedy algorithm, after a few iterations, the algorithm starts wasting a lot of time looking through all origin-destination pairs and checking whether any extra flow can be routed for each of them.

In our experiments, we also tested a modification of Greedy algorithm, where the augmenting path is chosen as *the longest* shortest path over all origin-destination pairs. Interestingly, it turned out that this "reversed" Greedy algorithm yields similar results to the "classical" one in terms of precision and CPU time.

	Greedy		FPTAS	
	CPU time	ω'	CPU time	ω'
L = 6	0.717	0.3	263	0.08
L = 9	0.245	0.038	402	0.004
L = 12	0.225	0.017	592	0.004

Table 6: Average CPU times (sec.) and a posteriori estimated approximation ω' on 300 instances for $\omega = 0.2$ and different values of L

6. FURTHER RESEARCH

We expect that an implementation of the FPTAS may be improved using the line search method for updating the current primal and dual solutions as proposed in [1] and by the means of parallel computations. The exact LP-model discussed in Section 2 is based on a Kirchoff-type formulation in an extended graph. An alternative column generation approach, assuming that columns are the length-bounded paths, has no polynomial time-bound but is often efficient in practice (see e.g. [20, 22]), so it would be interesting to make an experimental comparison to it. Using a generic CPLEX solver may not be the best option for the LP model considered in this paper, so further research might include a comparison of the algorithms presented here with some specialized optimization methods for the problems with multiflows-like structures, e.g. from [7, 8, 12]. Comparison of different greedy rules also deserves a further research, as it was done e.g. in [2] for the Minimum Cost Multiple-source Unsplittable Flow Problem.

7. CONCLUSIONS

We have considered three approaches to solving the length-bounded maximum multicommodity flow problem with unit edge-lengths. Following the approach of Garg and Könemann [13] with an improvement from [11], we have developed an FPTAS for this problem. The proposed FPTAS has a lower time complexity bound compared to the previously known algorithms, designed for problems with the length functions of more general form. We have also implemented a heuristic algorithm based on the well-known greedy approach, which in our case consists in augmenting the multicommodity flow via shortest paths. The third approach considered in the paper yields exact solutions by means of CPLEX solver applied to an LP model on the time-expanded network from [21].

The FPTAS and Greedy heuristic proposed in this paper turned out to be significantly faster than the CPLEX LP solver, especially on the instances with large networks and great number of demands. The FPTAS is more accurate but requires more CPU time than Greedy, which may be a decisive factor in practical applications. In general, the practical precision of FPTAS turns out to be much better than the theoretical guarantee.

Acknowledgement: This research is supported by the Russian Science Foundation grant 15-11-10009.

REFERENCES

[1] Albrecht, Ch., "Global routing by new approximation algorithms for multicommodity flow", *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 20 (5) (2001) 622–632.

[2] Asano, Y., *Experimental Evaluation of Approximation Algorithms for the Minimum Cost Multiple-source Unsplittable Flow Problem*, in: ICALP Satellite Workshops, Carleton Scientific, Waterloo, Ontario, Canada, 2000, 111–122.

[3] Baier, G., *Flows with Path Restrictions*, Ph.D. Diss., TU Berlin, Berlin, 2003.

[4] Bienstock, D., *"Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice"*, International Series in Operations Research & Management Science, Springer, New York, USA, 2002.
[5] Ben-Ameur, W., "Constrained-length connectivity and survivable networks", *Networks*, 36 (1) (2000) 17–33.
[6] Baier, G., Erlebach, T., Hall, A., Köhler, T., Kolman, P., Pangrác, O., Schilling, H., and Skutella, M., "Length-bounded cuts and flows", *ACM Trans. Algorithms*, 7 (1) (2010) 4:1–4:27.
[7] Castro, J., "Solving difficult multicommodity problems with a specialized interior-point algorithm", *Annals of Operations Research*, 124 (2003) 35–48.
[8] Castro, J., "Interior-point solver for convex separable block-angular problems", *Optimization Methods and Software*, 31 (2016) 88-109.
[9] Chaudhuri, K., Papadimitriou, C., and Rao, S. *"Optimum routing with quality of service constraints"*, *Unpublished manuscript*, 2004.
[10] Dinic, E.A., "Algorithm for solution of a problem of maximum flow in networks with power estimation", *Soviet Mathematical Docladi*, 11 (1970) 1277–1280.
[11] Fleischer, L.K., "Approximating fractional multicommodity flow independent of the number of commodities", *SIAM J.Disc.Math.*, 13 (2000) 505–520.
[12] Frangioni, A., Gallo, G., "A bundle type dual-ascent approach to linear multicommodity min cost flow problems", *INFORMS Journal On Computing*, 11 (1999) 370–393.
[13] Garg, N. and Könemann, J., "Faster and simpler algorithms for multicommodity flow and other fractional packing problems", in *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS'98)*, IEEE CS Press, Palo Alto, California, USA, 1998, 300–309.
[14] Goldberg, A.V., Oldham, A.D., Plotkin, S., and Stein, C., "An implementation of an approximation algorithm for minimum-cost multicommodity flows" in: *Proceedings of the 6-th Integer Programming and Combinatorial Optimization*, LNCS Vol. 1412, Berlin, Springer, 1998, 338–352.
[15] Garg, N., Vazirani, V., and Yannakakis, M., "Primal-dual approximation algorithms for integral flow and multicut in trees", *Algorithmica*, 18 (1997) 3–20.
[16] Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., and Yannakakis, M., "Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems", *Proceedings of the 31st Annual ACM Symposium on Theory of Computing, ACM,*Atlanta, GA, USA, 1999, 19–28.
[17] Hu, T.C., *Integer Programming and Network Flows*, Addison-Wesley Publishing Company, Reading, MA, 1970.
[18] Kleinberg, J. M., *Approximation algorithms for disjoint paths problems*, Ph.D. Diss., MIT, Cambridge, MA, 1996.
[19] Kolliopoulos, S. G., *Exact and approximation algorithms for network flow and disjoint-path problems*, Ph.D. Diss., Dartmouth College, Hanover, NH, 1998.
[20] Belaidouni, M. and Ben-Ameur, W., "On the minimum cost multiple-source unsplittable flow problem", *RAIRO Oper. Res.* 41 (3) (2007) 253–273.
[21] Kolman, P. and Scheideler, C., "Improved bounds for the unsplittable flow problem", *J. Algor.*, 61 (1) (2006) 20–44.
[22] Malashenko, Yu. E. and Stanevichyus, I.A., "The solution of the multi-product problem with integer-valued flows", *USSR Computational Mathematics and Mathematical Physics*, 22 (3) (1982) 245-249.
[23] Nemhauser, G.L. and Wolsey, L.A., *Integer and Combinatorial Optimization*, Wiley-Interscience, New York, NY, 1988.
[24] Van Mieghem, P., Kuipers, F.A., Korkmaz, T., Krunz, M., Curado, M., Monteiro, E., Masip-Bruin, X., Sole-Pareta, J. and Sanchez-Lopez, S., "Quality of service routing", *Quality of Future Internet Services, LNCS,* Springer, Berlin, (2856) 2003, 80–117.
[25] Radzik, T., "Experimental study of a solution method for multicommodity flow problems", in *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments*, San Francisco, CA, USA, 2000, 79–102.
[26] Schrijver, A., *Combinatorial Optimization. Polyhedra and Efficiency*, Vol. C, Springer, 2003.
[27] Tang, Z., Zhao, B., Yu, W., Feng, Z., and Wu, C., *"Software defined satellite networks: Benefits and challenges"*, Proceedings of Computing, Communications and IT Applications Conference (ComComAp), IEEE, Beijing, China, 2014, 127-132.

[28] Tardos, E., "A strongly polynomial algorithm to solve combinatorial linear programs", *Oper. Res.*, 34 (2) (1986) 250–256.
[29] Tsaggouris, G. and Zaroliagis, C., "Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications", *Theory Comput. Syst.*, 45 (1) (2009) 162–186.
[30] Villavicencio, J., "Solving multicommodity flow problems by an approximation scheme", *SIAM Journal on Optimization.*, 15 (4) (2005) 971 – 986.

APPENDIX

This appendix contains the proofs of Theorem 3.4, Lemma 3.2 and Lemma 3.3, omitted in Subsection 3.1.

Proof of Lemma 3.2. Initially, $\ell(e) = \delta$ for all edges e. The last time the length of an edge is updated, it is on a path of length less than one, and it is increased by at most a factor of $1 + \varepsilon$. Thus the final length of any edge is at most $1 + \varepsilon$. Since every augmentation increases the length of some edge by a factor of at least $1+\varepsilon$, the number of possible augmentations is at most $m \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$. □

Proof of Lemma 3.3. Every time the total flow on an edge e increases by a fraction $0 < a_j \le 1$ of $u(e)/\log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$, its length is multiplied by $1 + a_j\varepsilon$. Since $1 + a\varepsilon \ge (1 + \varepsilon)^a$ for all $0 \le a \le 1$, we have $\Pi_j(1 + a_j\varepsilon) \ge (1 + \varepsilon)^{\sum_j a_j}$, when $0 \le a_j \le 1$ for all j. So every time the flow on an edge increases by its capacity divided by $\log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$, the length of the edge increases by a factor of at least $1 + \varepsilon$. Initially $\ell(e) = \delta$ and at the end $\ell(e) < 1+\varepsilon$, so the total flow on e cannot exceed $u(e)$. □

Proof of Theorem 3.4. Part (i) follows from Lemma 3.2.

Now consider part (ii). Let ℓ_j denote the length function after j-th augmentation in Algorithm 3.1 and let $\alpha(\ell)$ denote the length of a shortest path in $\mathcal{P}(L)$ w.r.t. a length function ℓ. Given a length function ℓ, define $D(\ell) := \sum_e \ell(e)u(e)$, and let $D_j := D(\ell_j)$. Then D_j is the dual objective function value corresponding to ℓ_j and $\beta := \min_\ell D(\ell)/\alpha(\ell)$ is the optimal dual objective value. Let g_j be the primal objective function value after j-th augmentation and let P be the augmenting path. Denote $\sigma := \log_{1+\varepsilon} \frac{1+\varepsilon}{\delta}$. Then for each $j \ge 1$,

$$D_j = \sum_e \ell_j(e)u(e) = \sum_e \ell_{j-1}(e)u(e) + \varepsilon \sum_{e\in P} \ell_{j-1}(e)u$$

$$\le D_{j-1} + \varepsilon(g_j - g_{j-1})(1 + \varepsilon)\sigma\alpha(\ell_{j-1}),$$

which implies that

$$D_j \le D_0 + \varepsilon(1 + \varepsilon)\sigma \sum_{j'=1}^{j} (g_{j'} - g_{j'-1})\alpha(\ell_{j'-1}). \tag{16}$$

Consider the length function $\ell_j - \ell_0$. Note that $D(\ell_j - \ell_0) = D_j - D_0$. For any path used by the algorithm, the length of the path using ℓ_j versus $\ell_j - \ell_0$ differs by at

most δL. Since this holds for the shortest path using length function $\ell_j - \ell_0$, we have $\alpha(\ell_j - \ell_0) \geq \alpha(\ell_j) - \delta L$. Hence

$$\beta \leq \frac{D(\ell_j - \ell_0)}{\alpha(\ell_j - \ell_0)} \leq \frac{D_j - D_0}{\alpha(\ell_j) - \delta L}.$$

Using the bound on $D_j - D_0$ from equation (16), we obtain

$$\alpha(\ell_j) \leq \delta L + \frac{\varepsilon(1+\varepsilon)\sigma}{\beta} \sum_{j'=1}^{j} (g_{j'} - g_{j'-1})\alpha(\ell_{j'-1}).$$

Observe that, for fixed j, this right hand side is a non-decreasing function on $\alpha(\ell_0), \ldots, \alpha(\ell_{j-1})$. So, for any sequence of upper bounds $\alpha'_j,\ j = 1, \ldots,$ on $\alpha(\ell_j),\ j = 1, \ldots,$ we have

$$\alpha(\ell_j) \leq \alpha'_j = \alpha'_{j-1}(1 + \varepsilon(1+\varepsilon)\sigma(g_j - g_{j-1})/\beta) \leq \alpha'_{j-1} e^{\varepsilon(1+\varepsilon)\sigma(g_j - g_{j-1})/\beta},$$

where the last inequality uses the fact that $1 + a \leq e^a$ for $a \geq 0$. We can use a valid upper bound $\alpha'_0 = \delta L$, which implies

$$\alpha(\ell_j) \leq \delta L e^{\varepsilon(1+\varepsilon)\sigma g_j/\beta}.$$

By the stopping condition, after the last augmentation (let it be the augmentation number t) we have

$$1 \leq \alpha(\ell_t) \leq \delta L e^{\varepsilon(1+\varepsilon)\sigma g_t/\beta}$$

and hence

$$\frac{g_t}{\beta} \geq \frac{\ln(\delta L)^{-1}}{\varepsilon(1+\varepsilon)\sigma} = \frac{\ln(1+\varepsilon)\ln(L\delta)^{-1}}{\varepsilon(1+\varepsilon)\ln\frac{1+\varepsilon}{\delta}}.$$

Recalling that $\delta = \frac{1+\varepsilon}{\sqrt[\varepsilon]{(1+\varepsilon)L}}$ we obtain

$$\frac{g_t}{\beta} \geq \frac{(1-\varepsilon)\ln(1+\varepsilon)}{\varepsilon(1+\varepsilon)} \geq \frac{(1-\varepsilon)(\varepsilon - \varepsilon^2/2)}{\varepsilon(1+\varepsilon)} \geq \frac{(1-\varepsilon)^2}{1+\varepsilon},$$

which ensures the required flow value. The feasibility of the obtained solution follows from Lemma 3.3. □