Communication Papers of the Federated Conference on Computer Science and Information Systems pp. 29–36 DOI: 10.15439/2019F85 ISSN 2300-5963 ACSIS, Vol. 20

Exact and approximation algorithms for joint routing and flow rate optimization

Konstanty Junosza-Szaniawski
Warsaw University of Technology
Faculty of Mathematics and Information Science
ul. Koszykowa 75, 00-662 Warszawa, Poland
Email: k.szaniawski@mini.pw.edu.pl

Dariusz Nogalski
Military Communication Institute
C4I Systems Department
ul. Warszawska 22A, 05-130 Zegrze, Poland
Email: d.nogalski@wil.waw.pl

***Abstract*—This paper addresses the comparison of algorithms for a version of the Network Utility Maximization (NUM) problem. The joint formulation of routing and transmission rate control within the multi-user and single-path setting is assumed within the NUM. Since the problem is NP-hard, the efficient heuristics are designed, implemented and compared experimentally with other existing heuristics and exact linear programming solver. The linear approximation is applied for a nonlinear utility function. The results of the experiments demonstrate a trade-off between computing time and precision of goal value.**

***Index Terms*—Mathematical programming, network optimization problem, network utility maximization, rate allocation, congestion control, multi-rate computer networks, routing, NP-hard problem, heuristic algorithm**

I. Introduction

The Network Utility Maximization (NUM) problem was introduced by Kelly et al. in 1998 [10]. Within the problem a single route is associated with a user (constant to the problem) and the goal is to maximize aggregate utility of all user rates (variable to the problem). Such a basic NUM problem is known to be polynomial-time solvable [17]. We refer to such a problem as the *basic NUM*.

This article addresses an extended problem where both user rates and routes are variable to the problem. We refer to such a problem as the *extended NUM* problem or simply, the *NUM* problem. Wang et al. prove such a problem to be *NP-hard* [17]. As a consequence, efficient heuristics are needed for larger networks.

Such a joint formulation of the problem is useful since path selection determines link congestion and the problem can be solved *efficiently*. Efficiency is becoming a critical problem in many application areas, e.g. in multimedia WSN (Wireless Sensor Networks). Multimedia data producing wireless sensor nodes require higher bandwidth, higher energy usage, and strict quality of service (QoS) requirements [13]. Additionally, we are interested in techniques to achieve *fairness* of multi-stream allocations [10] [11] [3].

We designed and implemented several heuristics and compared their performance with other existing heuristics and the exact linear programming solver. Due to the fact that our utility function was nonlinear, we applied linear approximation within optimization procedure. The results of the experiments demonstrate a trade-off between computing time and precision of algorithms.

Since our approach is centralized, it cannot be used directly in multi-rate networks operating in a distributed way. However, there are certain applications where our heuristics can be found useful. Firstly, in applications where centralized network controllers are applied, e.g. SDN. Secondly, to estimate a gap between single and multiple path solutions [18].

II. Motivation and related work

A. The complexity of various versions of network utility maximization problem

The complexity of the basic and the extended NUM was stated in the introduction. However, for various problem modifications it may differ. The decision version of m-commodity flow problem with fixed rates and single-path setting is NP-complete [4]. It is proved by reduction from the decision version of the bin-packing problem. In [5] it is shown that even the decision version of two-commodity integral flow problem is NP-complete, by reduction from the SAT problem.

B. Distributed models

Since the work of Kelly et al. [10] was published, there have been many works on price-based distributed network utility maximizing methods. Chiang et al. [3] present the TCP/IP joint optimization example and solve it through distributed decomposition. Horizontal decomposition of TCP-AQM Congestion Control to share link capacities among competing users is described. Vertical decomposition of TCP/IP Joint Congestion Control and Routing is described.

C. Centralized control in SDN networks

Amokrane el al. [1] propose an online energy efficient flow-based routing approach for SDN controller, which allows for dynamic reconfiguration of existing flows (paths and rates). Huang et al. [8] solve utility-optimized flow-level bandwidth allocation in hybrid SDNs.

D. Importance of single-path routing

Single-path routing is still important from the practical point of view in the following areas: hybrid SDN networks [8], energy-aware networks [1] [9], IP networks [3] [10], MANETs (Mobile Ad hoc NETworks) [16] and WSNs [13]. It imposes a smaller overhead in implementation and is less expensive to support, as opposed to the multi-path [18]. Multi-path routing, as opposed to single-path routing has several disadvantages/challenges: data packet reordering, route maintenance, route discovery and a computational overhead.

E. Fairness and Efficiency

In literature there are various definitions of *fairness*. Two of them are most common: Max-min fairness [14], and proportional fairness [10] [14] [19] [11] [3]. The *proportional fair* vector comes as a solution to maximization of aggregate utility given by the equation (11) . Utility functions can be interpreted as the level of elasticity of application traffic, user satisfaction, optimality of resource allocation efficiency, as well as fairness [3]. If $\alpha = 0$, the NUM problem reduces to system throughput maximization. If $\alpha = 1$, proportional fairness among competing users is attained. And finally, if $\alpha = \infty$, then max-min fairness is achieved (Lee et al. [11]). The improvement in fairness is achieved by sacrificing the network efficiency. As the value of α increases, the total rate decreases. Our problem formulation assumes $\alpha = 1/2$ for heuristics testing purpose.

F. Multicommodity flow problem and its heuristics

Our problem belongs to a class of more general flow problems, namely the multicommodity flow problem (MFP) (Leighton et al. [12]). However, our formulation is single-path, as opposed to the multi-path formulation in MFP. Additionally, we do not assume a fixed demand for each source-destination pair, but express such a demand with utility function. MFP heuristics may be found in many works e.g. Garg and Konemann [7], Fleisher [6].

G. Heuristics for NUM with single-path routing

Drwal [4] defines Utility-Maximizing Network Design problem understood as joint optimization over rates and paths (single-path). The work introduces Maximum Spanning Tree based algorithm, which gives a solution not less than a factor $O(\frac{1}{m})$ of an optimal solution, where m is the number of pairs.

Wang et al. [18] introduce several upper-bounds to estimate the performance gap between multi- and single-path solutions of joint optimization of transmission rates and paths. They introduce two heuristics: vertex-projection and greedy-branch-and-bound.

III. Main results

In our paper we compared experimentally several heuristics to the NUM problem: MstNum (see alg 1), IterativeMstNum (see alg 2), LPRoundingNum (see alg 3), LPBestPathNum (see alg 4) and IterativeLPBestPathNum (see alg 5).

Our main contribution is the design of two heuristics: IterativeMstNum (an iterative procedure based on maximum spanning tree) and IterativeLPBestPathNum (an iterative procedure based on LP solver).

The other heuristics may be found in literature: MstNum [4], LPRoundingNum [4] [18] and LPBestPathNum [18].

All the algorithms are implemented and compared experimentally with MilpNum (see alg 6), which gives an optimal solution.

IV. Problem definition

Let's assume:

1) $G = (V, E)$ an undirected graph representing a network, where V is a set of nodes and E a set of edges (this model could also be defined for a directed graph)
2) $c : E \to [0, \infty)$ capacity function
3) S a set of users
4) $F_{st} = \{(s_k, t_k) \in V \times V : k \in S, \text{ and } s_k, t_k \in V\}$ a set of (source,target) pairs - demands for transmission
5) $u : S \times [0, \infty) \to [0, \infty)$ utility function
6) $u(k, x) = u_k(x)$ utility value of transmission rate x for the user $k \in S$

The problem is to find a set of pairs $\{(x_k, P_k) : k \in S\}$, where x_k is a non-negative transmission rate assigned to the user (2), P_k is a transmission path in the graph G from s_k to t_k, assignments do not exceed (altogether) the edge capacity (3) and rates are maximized (1). Such an extended NUM problem is a generalization of the basic NUM formulated by Kelly [10].

$$\max \sum_{k \in S} u_k(x_k) \tag{1}$$

For the purpose of the article we assume $\forall_{i,j,\in S}\ u_i = u_j$.

$$\forall_{k \in S}\ x_k \geq 0 \tag{2}$$

$$\forall_{\{u,v\} \in E(G)} \sum_{k,\text{ s.t. } \{u,v\} \in E(P_k)} x_k \leq c(u, v) \tag{3}$$

MILP formulation: In MILP each flow f_k takes a single path P_k from the source s_k to the target t_k. We introduce a real variable f_{kuv} which indicates the amount of flow f_k passing via edge (u, v). The variable f_{kuv} is defined to indicate $u \to v$ flow direction and f_{kvu} reverse. We rewrite the constraint (3) as (4). We set a requirement that the flow is balanced (5), except source and terminal nodes (6)(7).

$$\forall_{\{u,v\} \in E(G)} \sum_{k,\text{ s.t. } \{u,v\} \in E(P_k)} f_{kuv} + f_{kvu} \leq c(\{u, v\}) \tag{4}$$

$$\forall_{k \in S}\ \forall_{u \in V \setminus \{s_k, t_k\}} \sum_{v:\{v,u\} \in E(G)} f_{kvu} = \sum_{w:\{u,w\} \in E(G)} f_{kuw} \tag{5}$$

$$\forall_{k \in S} \sum_{w:\{s_k,w\} \in E(G)} f_{ks_kw} = x_k \tag{6}$$

$$\forall_{k \in S} \sum_{v:\{v,t_k\} \in E(G)} f_{kvt_k} = x_k \tag{7}$$

We introduce a binary variable y_{kuv} which indicates whether flow f_k is passing via edge (u, v) (8). The variable

y_{kuv} is defined to indicate $u \to v$ flow direction and y_{kvu} reverse.

$$\forall_{k \in S}\ \forall_{\{u,v\} \in E(G)}\ y_{kuv}, y_{kvu} \in \{0,1\} \tag{8}$$

We bind the variable y_{kuv} with the variable f_{kuv} using two additional constraints (9)(10).

$$\forall_{kuv: k \in S, \{u,v\} \in E(G)}\ f_{kuv} \le y_{kuv} \cdot c(\{u,v\}) \tag{9}$$

$$\forall_{kuv: k \in S, \{u,v\} \in E(G)}\ y_{kuv} \le f_{kuv} \cdot M \tag{10}$$

where M is a large number.

In practice, *fair* allocations of capacity are needed. This can be obtained by choosing the utility function properly. In literature [3] [10] [4] most common is a family of functions:

$$u_k(x_k) = \begin{cases} w_k \frac{1}{1-\alpha} x_k^{1-\alpha}, & \alpha > 0, \alpha \neq 1 \\ w_k log(x_k), & \alpha = 1 \end{cases} \tag{11}$$

For the experiment purpose, we have chosen $\alpha = 1/2$, which gives the function (12). We assume flow value to be non-negative (2).

$$\forall k \in S\ \ u_k(x) = u(x) = 2\sqrt{x} \tag{12}$$

We approximate the function (12) with three linear functions over range $x \in [0,1]$ (see fig 1). Such a choice stems from our assumption $\forall \{u,v\} \in E(G)\ c(u,v) = 1$. If links differ in terms of capacity, approximation function could range from 0 to max edge capacity. For the approximation purpose, an additional equation is introduced (14). The goal function (1) is rewritten as (13).

$$\max \sum_{k \in S} (a_1 \cdot t_{1,k} + a_2 \cdot t_{2,k} + a_3 \cdot t_{3,k}) \tag{13}$$

where $a_1 = 12.00, a_2 = 3.00, a_3 = 1.3(3)$

$$\forall_{k \in S}\ \ x_k = t_{1,k} + t_{2,k} + t_{3,k} \tag{14}$$

where $0 \le t_{1,k} \le 1/36$, $0 \le t_{2,k} \le 8/36$, and $0 \le t_{3,k}$

Relaxed LP formulation: We relax the y_{kuv} variable to allow the flow to split and flow via multiple paths (15). Such relaxation formulates the problem known in literature as Multi-Commodity Flow (MCF). Since this is a linear program, MCF can be solved in polynomial time [15].

$$\forall_{k \in S} \forall_{\{u,v\} \in E(G)}\ y_{kuv}, y_{kvu} \in [0,1] \tag{15}$$

In this NUM formulation all constraints { (2), (4), (5), (6), (7), (14) } hold except {(8),(9),(10)}, which are replaced with (15).

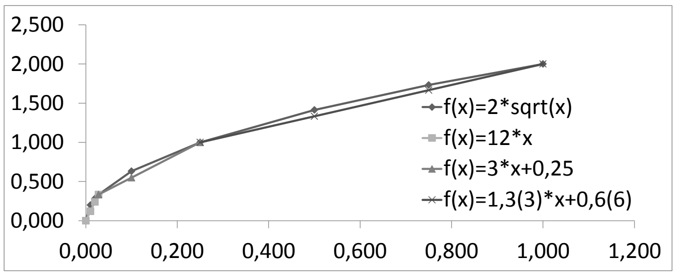

Fig. 1. Approximation of utility function $u_k(x) = 2\sqrt{x}$

V. Algorithms

To solve the NUM problem we have implemented the algorithms below. Some of them have been designed by us and some taken from literature (see chapter III Main results).

1) *MstNum(G, c, F_{st})* (see alg 1)
2) *IterativeMstNum(G, c, F_{st})* (see alg 2)
3) *LPRoundingNum(G, c, F_{st})* (see alg 3)
4) *LPBestPathNum(G, c, F_{st})* (see alg 4)
5) *IterativeLPBestPathNum(G, c, F_{st})* (see alg 5)
6) *MilpNum(G, c, F_{st})* (see alg 6)

All the algorithms assume the following *input* parameters:

- G graph representing a network
- c capacity function
- F_{st} a set of demands for transmission

and *output* parameters:

- FAS such a set of pairs $\{(x_k, P_k) : k \in S\}$ that constraints (2)(3) are satisfied
- U aggregate utility function value.

A. MST based NUM

The algorithm *MstNum* is based on [4]. It uses single MST (Maximum Spanning Tree) run (line 1). All flow paths belong to the MST (line 3). If competing flows pass the same edge, the edge capacity is shared among the flows proportionally with the use of *min* function (line 11).

In the worst case (having a network with $|E| = m$ parallel edges spanned between two vertexes), assuming $\forall_{e \in E} c(e) = k$, and $|S| = m$, the heuristics *MstNum* gives a ratio of a solution to an optimal solution equal to $1/m$. It is shown in [4] that the algorithm never returns a solution that is less than a factor $O(\frac{1}{m})$ of an optimal solution.

B. Iterative MST based NUM

The algorithm *IterativeMstNum* offers a large improvement over *MstNum* with respect to maximization of the goal function. It consists of two phases (init phase and iterative phase). The first phase (init) runs *MstNum* (line 1). The second phase runs an iterative utility improvement procedure (lines 3-28). Within each iteration (a single *while* loop), $|S|$ solutions are found (line 4-21), compared and the best one is selected (line 22) to produce the input flow allocation set (FAS) for the next iteration (line 26). To find an i-th solution (within an iteration) the algorithm produces i-th MST (line 16) taking c' as the

Algorithm 1 MstNum(G, c, F_{st})

```
1: $T = MST(G, c, F_{st})$
2: for $i \in S$ do
3:     Let $P_i$ be $s_i t_i$-path in T
4:     Update $FAS$ with $P_i$
5:     $x_i^{'} = min_{\{u,v\} \in E(P_i)} \; c(u, v)$
6:     $x_i = x_i^{'}$
7: end for
8: for $e \in E(T)$ do
9:     let $Shr_e = \{i \in S : e \in E(P_i)\}$
10:    if $(|Shr_e| > 1)$ then
11:        $\forall_{k \in Shr_e}$ let $x_k = min(x_k, \frac{x_k^{'}}{\sum_{\ell \in Shr_e} x_\ell^{'}} \cdot c(e))$
12:        Update $FAS$ with $x_k$
13:    end if
14: end for
15: return $(FAS, \sum_{k \in S} u(x_k))$
```

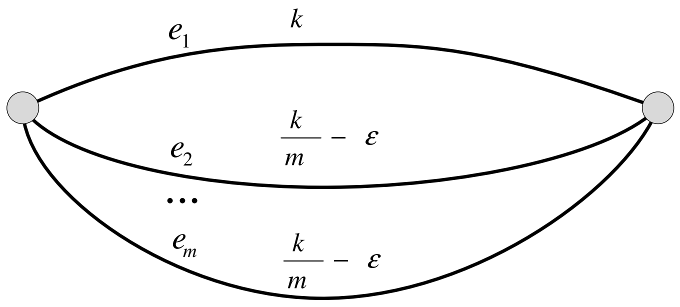

Fig. 2. Worst case for IterativeMstNum

Algorithm 2 IterativeMstNum(G, c, F_{st})

```
1: $FAS = MstNum(G, c, F_{st})$
2: $stop = false$
3: while $(stop == false)$ do
4:     for $i \in S$ do
5:         $(x_i^{'}, P_i^{'}) = FAS(i)$
6:         for $(u, v) \in V \times V$ do
7:             $c^{others}(u, v) = 0$
8:             for $j \in S$, s.t. $j \neq i$ do
9:                 $(x_j^{'}, P_j^{'}) = FAS(j)$
10:                if $(u, v) \in P_j^{'}$ then
11:                    $c^{others}(u, v) = c^{others}(u, v) + x_j^{'}$
12:                end if
13:            end for
14:            $c^{'}(u, v) = c(u, v) - c^{others}(u, v)$
15:        end for
16:        $T_i = MST(G, c^{'}, F_{st})$
17:        Let $P_i^{''}$ bet $s_i t_i$-path in $T_i$
18:        $x_i^{''} = min_{(u,v) \in P_i^{''}} \; c^{'}(u, v)$
19:        $\forall_{j \in S, j \neq i}$ recalculate $x_j^{''}$ using the idea given in
           alg 1 (lines 2-14)
20:        Compute $\Delta_i = u(x_i^{''}) - u(x_i^{'}) + \sum_{j \in S, j \neq i}(u(x_j^{''}) -
           u(x_j^{'}))$ using (12)
21:    end for
22:    $\Delta_{max} = max\{\Delta_i\}_{i \in S}$
23:    if $(\Delta_{max} \leq \Delta_\varepsilon)$ then
24:        $stop = true$
25:    else
26:        Update $FAS$ with the best solution in the current
           iteration $(x_{max}^{''}, P_{max}^{''})$
27:    end if
28: end while
29: return $(FAS, \sum_{k \in S} u(x_k))$
```

graph edge capacity (lines 6-15), which equals c decreased by the capacity used by the other users, and re-routes i-th flow to a possibly better location within the i-th MST (the new route may be the same as the previous one) (line 17). The Δ_i is computed as the aggregated sum of i-th flow utility difference and all j-th flow utility difference (line 20). The best solution is selected Δ_{max} (line 22). If the objective improvement is less or equal to Δ_ε, the stop condition is satisfied (line 24).

In the worst case (see fig 2), assuming $|S| = m$, a ratio of a solution to an optimal solution is determined by the equation (16). For $m \gg 1$ the ratio approximates to $1/2$. The algorithm *IterativeMstNum* never returns a solution that is less than a factor $O(\frac{1}{2})$ of an optimal solution.

$$\frac{sol}{opt} = \frac{1}{2 - 1/m} \tag{16}$$

However, for practical networks the heuristics exposes a solid ratio (see chapter VI Computational results).

C. Linear program relaxation and rounding NUM

The algorithm *LPRoundingNum* (a slight improvement of the one given in [4]) begins with solving linear program relaxation in phase1 (line 3). Subsequently in phase2 (lines 5-14) a rounding is done by random selection of the single path for each user. Finally, in phase3 (line 16) the LP problem is solved again taking fixed single paths identified in the previous phase as a constraint.

D. LP Best Path NUM

The algorithm *LPBestPathNum* begins with solving linear program relaxation (line 3). Within the relaxed LP solution, for each $k \in S$ the single max path (max min y_{kuv} on the path) is identified as P_k (line 6). The procedure is using the BFS (Breadth First Search) for that purpose. Finally, the LP problem is solved again taking fixed single paths identified (line 9) as a constraint. The algorithm is similar to the vertex projection method given in [18].

E. Iterative LP Best Path NUM

The algorithm *IterativeLPBestPathNum* constitutes $|S| + 1$ iterations. In each $\{1, .., |S|\}$ iteration, linear program relaxation is solved (line 7). Within the relaxed LP solution, for each $i \in S \setminus S'$ the single max path P_i (max min y_{iuv} on the path) is selected (line 10). Among the $|S \setminus S'|$-paths, the best (max) P_i is selected and added to the model (line 14) as

Algorithm 3 LPRoundingNum(G, c, F_{st})

```
1: Create a $problem$ with goal as (13)
2: Add constraints {(2),(4),(5),(6),(7),(14),(15)} to the $problem$
3: Solve the $problem$
4: Retrieve $\{y_{(k,u,v)\in S\times V\times V}\}$ and $U$ from the $problem$ solution
5: for $k \in S$ do
6:     Set vertex $u \in V$ s.t. $u = s_k$
7:     while $(u! = t_k)$ do
8:         Let's define a set $R_u = \{v_i\colon (u, v_i) \in V \times V$ and $y_{kuv_i} > 0\}$
9:         Let's define a probability function $Prob$ which assigns for each $v_i$ probability equal to $\frac{y_{kuv_i}}{\sum_{v_j \in R_u} y_{kuv_j}}$
10:        Choose randomly $v_i \in R_u$, where probability of selecting an element $v_i$ equals $Prob(v_i)$
11:        Add vertex $v_i$ to $P_k$
12:        $u = v_i$
13:    end while
14: end for
15: Add constraints $\{P_k : k \in S\}$ to the $problem$
16: Solve the $problem$
17: Retrieve $\{x_k : k \in S\}$ and $U$ from the $problem$ solution
18: Update $FAS$ with values $\{(x_k, P_k) : k \in S\}$
19: return $(FAS, \sum_{k\in S} u(x_k))$
```

Algorithm 4 LPBestPathNum(G, c, F_{st})

```
1: Create a $problem$ with goal as (13)
2: Add constraints {(2),(4),(5),(6),(7),(14),(15)} to the $problem$
3: Solve the $problem$
4: Retrieve $\{y_{(k,u,v)\in S\times V\times V}\}$ and $U$ from the $problem$ solution
5: for $k \in S$ do
6:     Using the BFS based procedure choose the max path $P_k$ with value $y_{P_k}$ (max min $y_{kuv}$ on the path)
7: end for
8: Add constraints $\{P_k : k \in S\}$ to the $problem$
9: Solve the $problem$
10: Retrieve $\{(x_k, P_k) : k \in S\}$ and $U$ from the $problem$ solution
11: Update $FAS$ with values $\{(x_k, P_k) : k \in S\}$
12: return $(FAS, \sum_{k\in S} u(x_k))$
```

a constraint (random selection, since there may be more than one best path (line 13)). The constraint forces i-flow to use only P_i edges in the next iterations.
In the last $|S|+1$ iteration the LP relaxation is solved assuming single fixed paths for all the $k \in S$ flows (line 17).

F. Mixed integer linear program NUM

The algorithm *MilpNum* is based on *MILP formulation* of the problem. It is a single phase algorithm (line 3).

Algorithm 5 IterativeLPBestPathNum(G, c, F_{st})

```
1: Create a $problem$ with goal as (13)
2: Add constraints {(2),(4),(5),(6),(7),(14),(15)} to the $problem$
3: Let's initiate a set of users with fixed single path $S' = \emptyset$
4: $stop = false$
5: while $(stop == false)$ do
6:     if $(S \setminus S'$ is not $\emptyset)$ then
7:         Solve the $problem$
8:         Retrieve $\{y_{(k,u,v)\in S\times V\times V}\}$ and $U$ from the $problem$ solution
9:         for $i \in S \setminus S'$ do
10:            Using the BFS based procedure choose the max path $P_i$ with value $y_{P_i}$ (max min $y_{iuv}$ on the path)
11:        end for
12:        Let's define a set $M$ as following $M = \{i \in S \setminus S'\colon y_{P_i} \cdot x_i == max\{y_{P_j} \cdot x_j\}_{j\in S\setminus S'}\}$
13:        Choose randomly $i \in M$, where probability of selecting an element $i$ equals $\frac{1}{|M|}$
14:        Add constraint $P_i$ to the $problem$
15:        $S' = S' \cup \{i\}$
16:    else
17:        Solve the $problem$
18:        $stop = true$
19:    end if
20: end while
21: Retrieve $\{(x_k, P_k) : k \in S\}$ and $U$ from the $problem$ solution
22: Update $FAS$ with values $\{(x_k, P_k) : k \in S\}$
23: return $(FAS, \sum_{k\in S} u(x_k))$
```

Algorithm 6 MilpNum(G, c, F_{st})

```
1: Create a $problem$ with goal as (13)
2: Add constraints {(2),(4),(5),(6),(7),(8),(9),(10),(14)} to the $problem$
3: Solve the $problem$
4: Retrieve $\{(x_k, P_k) : k \in S\}$ and $U$ from the $problem$ solution
5: Update $FAS$ with values $\{(x_k, P_k) : k \in S\}$
6: return $(FAS, \sum_{k\in S} u(x_k))$
```

VI. Computational results

A. Experiment Setup

The algorithms were programmed in Python 2.7 with the use of CPLEX Python library 12.8.0. The algorithm *MilpNum* uses CPLEX executable. All the simulations were run on a PC with 2-core 2.7GHz and 8GB RAM.

The experiments were conducted on five types of networks. The first network *Net7* with 7 nodes and 10 links [9]. The second network *Net22* with 22 nodes and 40 links [2]. In the second network nodes 1-8 and T (which is a network exit node) were source-destination nodes, whereas nodes 9-

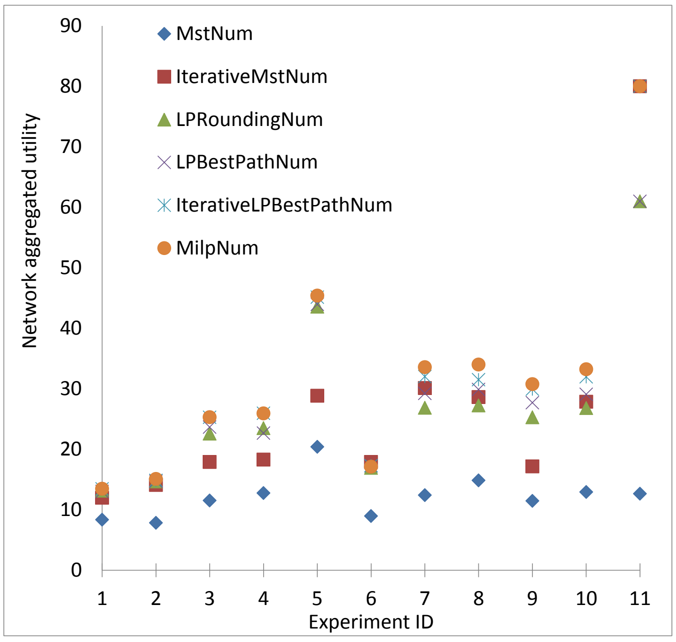

Fig. 3. Network aggregate utility (experiments $S1 - S11$)

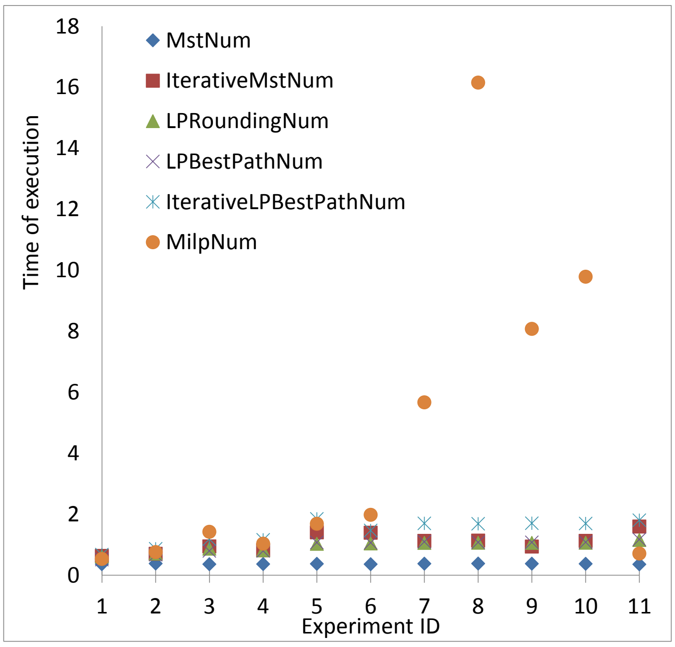

Fig. 4. Time of execution $\sqrt[4]{T}$ [sec.] (experiments $S1 - S11$)

21 were transit nodes. The third, grid network *Net64* with 8x8 nodes. The fourth network *Net42* with 40 disjoint paths spanned between node 1 and 42. Finally, the fifth, grid network *Net400* with 20x20 nodes. In all the networks above each link had a capacity equal to 1.

1) Verification of algorithm performance for different network structure and size: The experiment parameters were as follows: S1 - Net7, 10 pairs as given in [9]; S2 - Net22, 10 users randomly selected [1]; S3/4 - Net22, 20 users randomly selected [1], S5 - Net22, 50 users randomly selected [1]; S6 - Net64, 20 users: 10 x (1,64) and 10 x (8,57); S7/8 - Net64, 20 users randomly selected; S9/10 - Net64, 20 users randomly selected [2]; S11 - Net42, 20 users (1,42).

The algorithm *IterativeMstNum* was run with $\Delta_\varepsilon = 0.1$.

In each experiment, the algorithms: *LPRoundingNum*, *LPBestPathNum* and *IterativeLPBestPathNum* were run 20 times and the average values of objective U_{avg} and execution time T_{avg} are presented.

2) Verification of algorithm performance for different number of users: Additional 10 experiments (on Net64) were conducted to demonstrate an increase in utility and time upon an increase in the number of users: S12 - 10, S13 - 20, ... , S21 - 100 users [2].

3) Verification of algorithm performance for a large network: Finally, additional 3 experiments were conducted to capture utility and time for the Net400: S22 - 20, S23 - 50, S24 - 100 users [2].

[1] 50% of the pairs constitute internal traffic (between nodes 1-8), and 50% of the pairs constitute external traffic (between nodes 1-8 and T)

[2] Traffic sources and targets were located on the grid edge, and were randomly selected. Each smaller user set was a subset of a larger set.

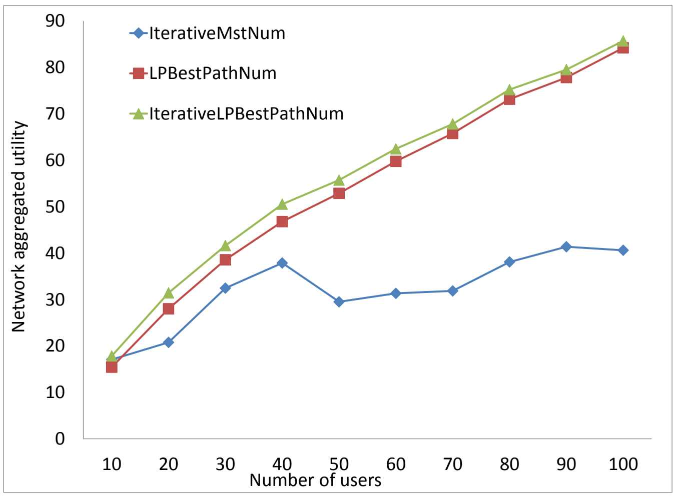

Fig. 5. Network aggregate utility in function of number of users (experiments $S12 - S21$)

B. *Performance of IterativeMstNum heuristics*

1) Different network structure and size - experiments $S1 - S11$: ***IterativeMstNum* vs *MstNum*.** The results from $S1 - S11$ demonstrate a large improvement in the objective value of the algorithm *IterativeMstNum* in comparison to *MstNum* (Fig. 3) - efficiency (utility) was even several times higher as high.

***IterativeMstNum* vs *LPRoundingNum*.** The utility values of *IterativeMstNum* observed in $S1 - 5, 9$ were not as high as the average ones achieved by *LPRoundingNum* (utility of *LPRoundingNum* was higher by $4\% - 51\%$ (Fig. 3)). On the other hand, in $S6 - 8, S10 - 11$ the results were in favor of *IterativeMstNum* (utility of *IterativeMstNum* was higher by

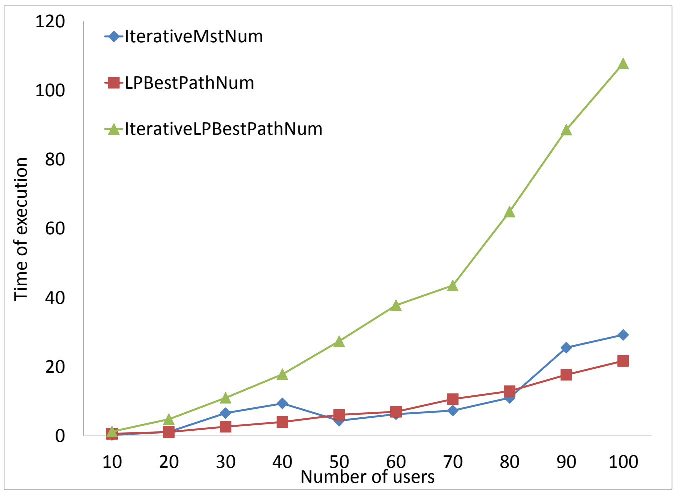

Fig. 6. Time of execution [sec.] in function of number of users (experiments $S12 - S21$)

TABLE I
UTILITY AND TIME FOR A LARGE NETWORK (EXPERIMENTS $S22 - S24$)

Experiment ID	Number of users	IterativeMstNum		LPBestPathNum		IterativeLPBestPathNum	
		U	$T[s]$	U_{avg}	$T_{avg}[s]$	U_{avg}	$T_{avg}[s]$
S22	20	33,36	10,96	31,54	27,09	36,18	139,78
S23	50	38,85	30,34	63,59	341,51	77,46	6152,44
S24	100	36,49	208,56	109,40	5129,93	-	-

$4\% - 31\%$ (Fig. 3)).

When it comes to the observed execution time, the algorithms *IterativeMstNum* and *LPRoundingNum* exposed similar performance, slightly in favour of *LPRoundingNum* (Fig. 4).

***IterativeMstNum* vs *MilpNum*.** The objective values of *IterativeMstNum* observed in the experiments $S6, S11$ were very close to the ones achieved by *MilpNum*. In the experiment $S6$ the value of *MilpNum*, lower than the value of *IterativeMstNum*, can be explained by an approximation error (Fig. 3). For a small number of users ($S1, S2$) the observed distance from the optimum (*MilpNum*) was not more than 12% , for a larger number of users ($S3 - S5, S7 - 10$) it ranged between 11% and 80%.

2) Different number of users - experiments $S12 - S21$: ***IterativeMstNum* vs *LPBestPathNum*.** In general, the observed utility of *IterativeMstNum* increased when the number of users grew, however the function was not monotonic. The utility was comparable with the one achieved by *LPBestPathNum* when the number of users was not high, up to 40 users for *Net64* (Fig. 5). Then, the utility difference was not higher than 35%. When the number of users was 10, for *Net64*, the utility of *IterativeMstNum* was higher by 11%. This number of users was correlated with the number of user-disjoint paths.

The time of execution of *IterativeMstNum* was comparable with the one achieved by *LPBestPathNum* regardless of the number of users (Fig. 6).

3) Large network - experiments $S22 - S24$: ***IterativeMstNum* vs *LPBestPathNum*.** The utility of *IterativeMstNum* was comparable with the one achieved by *LPBestPathNum* when the number of users was not high (TABLE I). When number of users was 20, for *Net400*, the utility of *IterativeMstNum* was higher by 5.8%. The same remark on user-disjoint paths applies, as above.

The time of execution of *IterativeMstNum* was much smaller than the one achieved by *LPBestPathNum* (an order of magnitude when the number of users was high - TABLE I).

C. Performance of LPBestPathNum and IterativeLPBestPathNum heuristics

1) Different network structure and size - experiments $S1 - S11$: ***LPBestPathNum* vs *LPRoundingNum*.** The average objective values of *LPBestPathNum* observed in the most of the experiments $S1 - S11$ were superior to *LPRoundingNum* by up to 9.7% (Fig. 3), except for $S1, S4$ - 1% and 3.6% opposite difference, respectively. Both heuristics demonstrated very similar execution time (Fig. 4).

***LPBestPathNum* vs *IterativeLPBestPathNum*.** In $S1 - S11$ the average observed objective values of *IterativeLPBestPathNum* were higher than values of *LPBestPathNum* by up to 14.5%, except for the parallel edge network (*Net42*) by 31% (Fig. 3). However, *IterativeLPBestPathNum* exposed higher execution time due to additional $|S| - 1$ iterations (Fig. 4).

***IterativeLPBestPathNum* vs *MilpNum*.** In $S1 - S6, S11$ the optimal objective values (*MilpNum*) were higher than *IterativeLPBestPathNum* solution by not more than 1.5%. In $S7, S9 - 10$ - by not more than 4.8%. In $S8$ - by 7.9%.

2) Different number of users - experiments $S12 - S21$: ***IterativeLPBestPathNum* vs *LPBestPathNum*.** In $S12 - S21$ the average objective values of *IterativeLPBestPathNum* and *LPBestPathNum* grew when the number of users grew. The utilities of *IterativeLPBestPathNum* were higher than those of *LPBestPathNum* (Fig. 5). The observed difference became smaller when the number of user grew. As observed, from 16% for 10 users to 2% for 100 users.

However, *IterativeLPBestPathNum* required much longer computation times than *LPBestPathNum*, with an increasing number of users (Fig. 6).

3) Large network - experiments $S22 - S24$: ***IterativeLPBestPathNum* vs *LPBestPathNum*.** The utility difference between *IterativeLPBestPathNum* and *LPBestPathNum* was more visible for large networks (20x20 nodes), as observed 14.7% (20 users) - 21.8% (50 users) in favor of the former (TABLE I). However, the computation time of *IterativeLPBestPathNum* for a large networks and large number of users may not be practically attainable (TABLE I).

D. Performance of MilpNum solver

The experiments $S7 - 10$ showed that *MilpNum* may expose a very long computation time to achieve an optimal solution, in the range of hours (Fig. 4). A possible, way around for *MilpNum* would be to configure the automatic termination when reaching a defined solution gap. As an example, in the experiment $S9$ *MilpNum* gave the value 26,22 (gap 16,95%) in 12s and 29 (5,75%) in 90s. IterativeLPBestPathNum gave 29,928 in 8,4s. In $S10$ *MilpNum* gave 30 (11,11%) in 12s and 31,66 (5%) in 268s. IterativeLPBestPathNum gave 31,962 in 8,2s.

VII. Conclusions

Since our problem is NP-hard, search for optimum (*MilpNum*) may not be practically attainable even for medium-sized grid networks (8x8 nodes) and a small number of users (~ 20).

The experiments show that the algorithm *LPBestPathNum* demonstrates a reasonably good trade-off between computing time and precision of the utility value.

As observed, our LP-based heuristics *IterativeLPBestPathNum* demonstrates up to $\sim 15\%$ higher utility, on networks with up to 64 nodes and 50 users, but it is slower, in comparison to *LPBestPathNum*. The difference in utility may be a few percent bigger for large grid networks (20x20 nodes) and a high number of users (~ 50), and even more for a special type of network - parallel edges. The bigger difference is caused by the edge allocation conflict. The *IterativeLPBestPathNum* reduces the probability of such conflicts by iterative path fixing (a single path fixed for each iteration).
The utility values of *IterativeLPBestPathNum*, on network with up to 64 nodes and 50 users, are lower than *MilpNum* by only $\sim 2\%$ on average.
The successful *IterativeLPBestPathNum* computations were conducted for grid networks with up to 20x20 nodes and 50 users. For comparison, the successful and not terminated *MilpNum* computations were conducted for grid networks with up to 8x8 nodes and 20 users.

As observed, our MST-based heuristics *IterativeMstNum* ensures reasonably good utility in certain cases. When the number of users is not high (the number is correlated with the number of user-disjoint paths), the *IterativeMstNum* utility is lower than *MilpNum* by not more than $\sim 10\%$. With such an assumption it has efficiency comparable to *LPBestPathNum*. When the number of users grows, the gap between *IterativeMstNum* and *LPBestPathNum* increases. The *IterativeMstNum* has utility which is equal to optimal for a special type of network - parallel edge network, since it avoids allocating the same edge to competing streams if alternative ones are available. In such networks, it may be much more efficient than *LPBestPathNum* (e.g. $S11 - 30\%$).
On average, for the observed cases, the *IterativeMstNum* utility value is lower than *MilpNum* by $\sim 25\%$.
Additionally, *IterativeMstNum* has computing time comparable to *LPBestPathNum* for small and medium-sized networks. For large networks and a high number of users, it may be an order of magnitude faster than *LPBestPathNum*. Finally, it does not use LP solver but a simple MST-based iterative procedure.

References

[1] A. Amokrane, R. Langar, R. Boutaba, and G. Pujolle. Flow-based management for energy efficient campus networks. *IEEE Transactions on Network and Service Management*, 12(4):565–579, Dec 2015. 10.1109/TNSM.2015.2501398.
[2] A.P. Bianzino, L. Chiaraviglio, M. Mellia, and J.L. Rougier. Grida: Green distributed algorithm for energy-efficient ip backbone networks. *Computer Networks*, 56(14):3219–3232, September 2012. https://doi.org/10.1016/j.comnet.2012.06.011.
[3] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, Jan 2007. 10.1109/JPROC.2006.887322.
[4] M. Drwal. Approximation algorithms for utility-maximizing network design problem. In Henry Selvaraj et al., editor, *Progress in Systems Engineering: Proceedings of the Twenty-Third International Conference on Systems Engineering*, volume 366 of *Advances in Intelligent Systems and Computing*. Springer International Publishing, Cham, 2015. https://doi.org/10.1007/978-3-319-08422-0-60.
[5] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In IEEE, editor, *16th Annual Symposium on Foundations of Computer Science 13-15 October 1975*, October 1975. 10.1109/SFCS.1975.21.
[6] L. K. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 24–31, Oct 1999. 10.1109/SFFCS.1999.814573.
[7] N. Garg and J. Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280)*, pages 300–309, Nov 1998. 10.1109/SFCS.1998.743463.
[8] X. Huang, T. Yuan, and M. Ma. Utility-optimized flow-level bandwidth allocation in hybrid sdns. *IEEE Access*, 6:20279–20290, 2018. 10.1109/ACCESS.2018.2820682.
[9] P. Jaskóła, P. Arabas, and A. Karbowski. Simultaneous routing and flow rate optimization in energy-aware computer networks. *International Journal of Applied Mathematics and Computer Science*, 26(1):231–243, 2016. 10.1515/amcs-2016-0016.
[10] F.P. Kelly, A.K. Maulloo, and D.K.H. Tan. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, March 1998. https://doi.org/10.1057/palgrave.jors.2600523.
[11] J.W. Lee, M. Chiang, and R. A. Calderbank. Jointly optimal congestion and contention control based on network utility maximization. *IEEE Communications Letters*, 10(3):216–218, March 2006. 10.1109/LCOMM.2006.1603389.
[12] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999.
[13] M. Macit, V. C. Gungor, and G. Tuna. Comparison of qos-aware single-path vs. multi-path routing protocols for image transmission in wireless multimedia sensor networks. *Ad Hoc Networks*, 19:132 – 141, 2014. https://doi.org/10.1016/j.adhoc.2014.02.008.
[14] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, Oct 2000. 10.1109/90.879343.
[15] M. E. Pfetsch and Ch. Liebchen. Multicommodity flows and column generation. In *Lecture Notes*. Zuse Institute Berlin, 2006.
[16] P. R. Satav and P. M. Jawandhiya. Review on single-path multi-path routing protocol in manet: A study. In *2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–7, Dec 2016. 10.1109/ICRAIE.2016.7939514.
[17] J. Wang, L. Li, S.H. Low, and J.C. Doyle. Cross-layer optimization in tcp/ip networks. *IEEE/ACM Transactions on Networking*, 13(3):582 – 595, June 2005. 10.1109/TNET.2005.850219.
[18] M. Wang, C. W. Tan, W. Xu, and A. Tang. Cost of not splitting in routing: Characterization and estimation. *IEEE/ACM Transactions on Networking*, 19(6):1849–1859, Dec 2011. 10.1109/TNET.2011.2150761.
[19] W. Wang, M. Palaniswami, and S. H. Low. Application-oriented flow control: Fundamentals, algorithms and fairness. *IEEE/ACM Transactions on Networking*, 14(6):1282–1291, Dec 2006. 10.1109/TNET.2006.886318.