Heuristics for Max-Throughput Single-Path Netowork Routing subject to Max-Min Fair Flow Allocation

Lorenzo Gentile

December 16, 2014

Abstract

I propose two heuristics to solve the following network routing problem: given a graph with arc capacities, a set of commodities with a fixed origin-destination pair, it is required to route each commodity over a single path so as to maximize the throughput, subject to the constraint that flows are allocated according to the Max-Min Fair (MMF) principle. In such paradigm, the commodity with the smallest allocation is maximized and also, in turn, the second smallest, the third smallest and so on. The concept of best-effort service in the Internet can be associated to that of MMF, since the network is expected to provide the best possible service in terms of rate without privileging any specific flow.

1 Introduction

Consider the task of sharing the network capacity (bandwidth) among n commodities (users). Let $\phi \in \Re^3$ denote a flow vector where the i-th component ϕ_i corresponds to the flow allocated to the i-th commodity. Let σ be a sorting operator permuting the components of ϕ in non decreasing order, i.e., such that $\sigma(\phi)_i \leq \sigma(\phi)_j$ whenever $i < j$.

Definition: A flow vector $\phi \in \Re^3$ is $Max - Min\ Fair$ (MMF) if, for any other flow vector $\phi' \in \Re^3$, $\sigma(\phi)$ lexicographically dominates $\sigma(\phi')$, i.e., either $\sigma(\phi) = \sigma(\phi')$ or there exists an integer k, with $1 \leq k \leq n$, such that $\sigma(\phi)_k > \sigma(\phi')_k$ and $\sigma(\phi)_l = \sigma(\phi')_l$ for all $l < k$.

In other words, a flow vector is MMF if there is no way to increase the flow of any commodity without decreasing the flow of a commodity with a smaller or equal flow. When the routing paths are given, a simple polynomial-time algorithm, the so-called $Water\ Filling$ algorithm yields an MMF flow allocation [1]. Such algorithm is used by the proposed heuristics.

It is now possible to define formally the problem:

Max-Throughput Single-Path Network Routing subject to MMF flow allocation (MT-SPNR-MMF): Given a directed graph $G = (V, A)$ with capacities $c_{i,j}$ for each $(i, j)\ \epsilon\ A$ and a set of commodities K with origin-destination pairs (s, t), route each commodity over a single path so as to maximize the throughput, subject to the constraint that the amount of flow allocated to the commodities be MMF w.r.t. all the flow allocations that are feasible for the paths that have been chosen.
MT-SPNR-MMF is a bilevel problem where, at the upper level, the leader (network operator) selects a routing for each commodity and, at the lower level a follower (TCP protocol) allocates the flows to the chosen paths according to the MMF paradigm.
Note that the solution of that problem is MMF given the paths chosen by the network operator. It is not MMF, in general, w.r.t. all the possible paths.
The objective of the network operator is maximizing the throughput by choosing appropriate paths, given the unavoidable constraint of TCP protocol that flows are allocated according to MMF.
Maximing the throughput and finding a solution that is MMF w.r.t. all possible paths are, in general, contrasting targets.
The proposed heuristic is justified by the fact that the MT-SPNR-MMF problem is $NP - hard$ [2].

2 Heuristics

The heuristics receive as input a directed graph $G = (V, A)$, a set of commodities K with origin-destination pairs (s, t) and an integer T that represents the number of times the heuristc tries to solve the instance. After T attempts, the heuristic will return the best solution found. The term ϵ used in line 12 is just a small positive number (e.g. 0.001) in order to avoid division by 0 when the capacity of the arc is saturated. In such a situation the cost of the arc is very high and, it could be selected for routing a commodity just in case there are not other paths available.

Algorithm 1 showed best performance w.r.t other solutions that were different in the following points:

- Order in which commodities are considered for calculating a path (line 6)
- Choice of a path for the commodities (line 9)
- Cost function for the arcs (line 12).

Attempts that brought to Algorithm 1 are described briefly in subsection 2.1. In subsection 2.2 is described a simple variation of Algorithm 1 that uses a local search approach. Such variation showed slightly better perfomances w.r.t.

Algorithm 1.

```
   Data: G = (V,A), K, T
   Result: Pbest
   /* lists that will contain the candidate and the best
      solution, i.e.  the paths and their flows                */
 1  Pcandidate = [];
 2  Pbest =[];
   /* each path is composed by a list of arcs, each arc is
      characterized by a capacity, a flow and a cost           */
 3  for a in A do
 4  |  a.cost = 1/a.capacity;
 5  end
 6  for t in range(0,T) do
    |  /* shuffle K in order to differentiate the solutions
    |     calculated during T iterations                       */
 7  |  shuffle(K);
 8  |  Pcandidate.clear();
 9  |  for k in K do
    |  |  /* calculate the shortest path for the commodoty k and
    |  |     add it to Pcandidate                              */
10  |  |  Pcandidate.append(shortestpath(k));
    |  |  /* calculate MMF flows for paths contained in
    |  |     Pcandidate                                        */
11  |  |  waterfilling(Pcandidate);
    |  |  // update cost of each arc according to his flow
12  |  |  for a in A s.t. belongs to at least one path in Pcandidate do
13  |  |  |  a.cost = 1/(a.capacity - a.flow + ϵ);
14  |  |  end
15  |  end
    |  // update Pbest if Pcandidate is better
16  |  if Pcandidate.flow > Pbest.flow then
17  |  |  Pbest = Pcandidate;
18  |  end
19 end
```

Algorithm 1: Multi-Start greedy algorithm with randomized order of paths selection.

2.1 Attempts to find a good heuristic

Order in which commodities are considered for calculating a path: For each origin-destination pair $(s,t)\ \epsilon\ K$, the minimum number of arcs that have to be removed in order to make t not reachable from s has been calculated. This quantity, that will be called minimum edge cut, is a lower bound of all the possible paths from s to t. Before obtaining Algorithm 1, commodities were considered:

- In non-decreasing order of minimum edge cut
- The origin-destination pair with minimum edge cut as first, the others randomly.

Finding a path first for the origin-destination pairs (s,t) for which it is not guaranteed there are many possible paths from s to t, was an attempt to take as soon as possible the obliged decisions (e.g. select a path when it is the only one possible from an origin to a destination).
As said before, minimum edge cut is just a lower bound and this approach didn't work very well. Considering commodities in a totally random order allowed to explore more extensively the space of the solutions, and then getting a better perfomance.

Choice of a path for the commodities: A GRASP approach has been used. Instead of selecting the shortest path for each commodity at line 10, a path was selected randomly between the k shortest paths.

Cost function for the arcs: The following cost functions have been used:

- a.cost = (1 + a.overlap)/(a.capacity - a.flow + ϵ)
- a.cost = 1/(a.capacity - random.uniform(0.5,1)*a.flow + ϵ)

Where a.overlap represents the number of paths using the considered arc.

The first cost function is an attempt to restrain overlapping between paths.
The second cost function allows the heuristic to select occasionally an arc also if its residual capacity is lower w.r.t. some other arc that could be used to permit comunication between the considered origin-destination pair.

2.2 A local search approach

Algorithm 2 exploits solutions found by the heuristic during previous attempts in order to build a new solution at the current attempt.

It is charterized by two more parameters w.r.t Algorithm 1:

- NREUSE: Minimum number of times a good solution is eploited in order to buil a new solution
- REUSEPERCENTAGE: Percentage of the paths of a good solution that are reused (taken as explained below).

With good solution it is intended a solution found at attempt t that is better than the best solution found at some attempt $t^{'} < t$.
In order to exploit a good solution, the first REUSEPERCENTAGE of the paths in Pbest are used (line 10) to build a new solution for at least NREUSE times. If after NREUSE times a new best solution has not been found, then Pcandidate will be empty and, a new solution will be built from the scratch.

```
Data: G = (V,A), K, T, NREUSE, REUSEPERCENTAGE
Result: Pbest
Pcandidate = [];
Pbest =[];
/* number of times a REUSEPERCENTAGE of a good solution is
   reused                                                    */
 reuse = 0;
for a in A do
    a.cost = 1/a.capacity;
end
for t in range(0,T) do
    shuffle(K);
    if reuse > 0 then
        // first REUSEPERCENTAGE of the solution is reused
         Pcandidate = Pbest[0:int(len(Pbest)*REUSEPERCENTAGE)]
    else
        Pcandidate.clear();
    end
    for k in K s.t. there is not a path dedicated in Pcandidate do
        Pcandidate.append(shortestpath(k));
        waterfilling(Pcandidate);
        for a in A s.t. belongs to at least one path in Pcandidate do
            a.cost = 1/(a.capacity - a.flow + ϵ);
        end
    end
    if Pcandidate.flow > Pbest.flow then
        Pbest = Pcandidate;
        /* set reuse = NREUSE excepting when t = 0 (when t = 0
           Pcandidate.flow > Pbest.flow in any case, it is not
           worth reusing the first solution found)           */
        if t > 0 then
            reuse = NREUSE;
        end
    else
        /* decrement reuse if the candidate solution is worse
           than the best                                     */
        if reuse > 0 then
            reuse = reuse -1;
        end
    end
end
```

Algorithm 2: Multi-Start greedy algorithm with randomized order of paths selection and occasional partial deletion of the solution.

3 Some computational results

I summarize the performances (w.r.t. LB and UB given respectively by a branch and bound with time limit of 1 hour and the continuous relaxation) of Algorithm 1 (Table 1) and Algrithm 2 (Table 2) on a set of 184 instances characterized by:

- In average 20.52 nodes and 30.33 commodities
- A maximum of 50 nodes and 72 commodities
- A minimum of 12 nodes and 3 commodities.

Note that the LB is the best solution found in 1 hour of computation. In some case it could be the optimal solution.
Both Algorithm 1 and Algorithm 2 obtain, in less than 10 minutes, solutions with a value that is (in average on the 184 instances) greater than 90 % of LB.

The algorithms have been implemented using Python and a graph library (completely implemented in Python) called NetoworkX.
Time perfomances could be improved further by implementing the algorithms using more efficient languages or libraries with Python bindings (e.g. graph-tool, igraph).

T	10	100	200	500
avg(100*a1/LB)	91.97	94.5	95	95.53
avg(100*a1/UB)	89.79	92.25	92.75	93.24
avg(100*(UB-a1)/a1)	12.1	8.99	8.36	7.76
avg(100*(UB-a1)/UB)	10.21	7.75	7.25	6.76
dev.st(100*a1/LB)	5.82	5.33	5.24	5.13
min(100*a1/LB)	73.47	73.47	73.47	73.47
perc. num > 80 (100*a1/LB)	98.38	98.92	98.92	98.92
perc. num > 90 (100*a1/LB)	62.16	78.37	81.62	83.78
time (min)	8.84	92.36	175.64	434.57

Table 1: Performance of Algorithm 1.

T	10	100	200	500
NREUSE	3	7	9	15
REUSEPERCENTAGE	0.5	0.5	0.5	0.7
avg(100*a2/LB)	92.2	94.78	95.35	95.83
avg(100*a2/UB)	89.99	92.51	93.06	93.51
avg(100*(UB-a2)/a2)	11.85	8.65	7.99	7.41
avg(100*(UB-a2)/UB)	10.01	7.49	6.94	6.49
dev.st(100*a2/LB)	5.74	5.3	5.21	5.1
min(100*a2/LB)	73.47	73.47	73.47	73.47
perc. num > 80 (100*a2/LB)	97.84	98.92	98.92	98.92
perc. num > 90 (100*a2/LB)	63.78	81.08	84.86	87.57
time (min)	9.01	90.51	178.48	446.13

Table 2: Performance of Algorithm 2.

In Table 3 are reported the perfomances of Algorithm 2 with T = 200 using different values of NREUSE and REUSEPERCENTAGE. The best performance has been obtained using NREUSE = 9 and REUSEPERCENTAGE = 0.5.
By increasing NREUSE and REUSEPERCENTAGE local search is privileged w.r.t. exploration of the space of the solutions.
Computational experiments suggest that Algorithm 2 works better when parameters are settled in order to get a good compromise between local search and exploration.

NREUSE	0	4	**9**	20	30	4	9	20	30
REUSEPERCENTAGE	0	0.5	**0.5**	0.5	0.5	0.9	0.9	0.9	0.9
avg(100*a2/LB)	95	95.25	95.35	95.33	95.18	95.01	95.11	94.8	94.58
avg(100*a2/UB)	92.75	92.96	93.06	93.04	92.91	92.73	92.82	92.53	92.32
avg(100*(UB-a2)/a2)	8.36	8.08	7.99	8.01	8.17	8.37	8.25	8.63	8.87
avg(100*(UB-a2)/UB)	7.25	7.04	6.94	6.96	7.09	7.27	7.18	7.47	7.68
dev.st(100*a2/LB)	5.24	5.19	5.21	5.21	5.16	5.25	5.3	5.33	5.19
min(100*a2/LB)	73.47	73.47	73.47	73.47	73.47	73.47	73.47	73.47	73.47
perc. num $>$ 80 (100*a2/LB)	98.92	98.92	98.92	98.92	98.92	98.92	98.92	98.92	98.92
perc. num $>$ 90 (100*a2/LB)	81.62	85.41	84.86	82.7	83.24	82.16	84.86	80	80.54
time (min)	175.64	185.09	178.48	196.27	178.14	187.37	185.94	181.3	180.57

Table 3: Parameters settings of Algorithm 2 with T = 200.

4 Conclusion and future work

I have proposed two heuristics for maximizing the throughput that a network operator can offer overall to his customers.
Both heuristics find, in less than 10 minutes, solutions with a value that is (in average on 184 testing instances) greater than 90 % of the value of the best solutions found using a branch and bound with time limit of 1 hour for each instance.
One possible direction for future work is using a *tabu search* in order to improve the best solution found by these heuristics.
Another possible direction is using a *genetic* approach so as to build new solutions using best solutions found in the past.

5 References

[1] D. Bertsekas and R. Gallager. *"Data Networks, 1992."*, Prentice-Hall, 1992.

[2] Edoardo Amaldi, Stefano Coniglio, Luca G. Gianoli, Can Umut Ileri. *"On single-path network routing subject to max-min fair flow allocation"*, 2013.