Fanout Inference From Link Count

Cédric Fortuny
LAAS-CNRS
7, avenue du Colonel Roche
31077 Toulouse, France
cfortuny@laas.fr

Olivier Brun
LAAS -CNRS
brun@laas.fr

Jean-Marie Garcia
QoSDesign
6, av Marcel Doret
31500 Toulouse, France
jmg@qosdesign.org

Abstract

The traffic matrix is the fundamental input data in network planning, simulation and traffic engineering. However, it is often unknown and its direct measurement with devices such as Netflow is a too heavy process for large high-speed networks. The estimation of the traffic matrix from link counts appears as the best alternative approach. Recent works assume that prior information do not allow alone an accurate estimation of the traffic matrix so they are using Netflow measures to calibrate their model. These models assume spatial and temporal relations between different instants of measure. We show in this paper that we can obtain similar error rates without this costly calibration phase thanks to a spatial and temporal relation introduced in [6].

1 Introduction

In the current competitive context, network operators have to constantly improve their networks to cope with the sustained increase in traffic and the new Quality of Service (QoS) requirements. In order to avoid network congestion and service degradation, a cost-efficient alternative to installing excessive amounts of capacity is to perform network and traffic engineering. To this end, a broad range of tasks have to be performed (e.g. route optimisation, network simulation) which all requires the same input data: the traffic matrix of the network. This matrix contains the volume of traffic exchanged between each pair of Origin/Destination (OD) in the network. However, this traffic matrix cannot be directly measured in large high-speed networks due to the high processing overhead introduced by current metrology tools. Netflow [2] is the most popular of such tools, but it is well-known that using it in large high-speed networks puts a high processing overhead on CPU of routers and generates a large amount of reporting traffic. These two reasons make these metrology tools unusable by network providers.

It is now broadly admitted that other measures have to be used for traffic matrix estimation. Lighter measures are obtained by SNMP which gives the traffic on each link of the network averaged over five minutes. Let N be the number of network nodes (so that there are $c = N(N-1)$ OD) and let L be the number of links. Let $\mathbf{X}$ be the c-vector of OD traffics and $\mathbf{Y}$ be the L-vector of link loads. Then the following holds,

$$\mathbf{Y} = A\mathbf{X} \tag{1}$$

Where A is the routing matrix: $A_{k,l}$ is the proportion of the demand for OD pair k which is routed on link l. This matrix is assumed to be known and constant during each measurement, so the traffic matrix estimation problem amounts to to find $\mathbf{X}$ from $\mathbf{Y}$ and A. This problem is under-constrained because in the usual networks, there are far more less links than OD pairs ($l << c$). Therefore, to solve this ill-posed problem, additional information is required.

We develop in this paper our contribution which is an estimation of the traffics without any Netflow measurement. To do that, we will estimate the fanout : a set of values ($p_{i,j}$) instead of the traffics. $p_{i,j}$ is in fact the ratio between the traffic from i to j and the sum of all the traffics leaving the node i. We assume these ratios are quite constant under certain conditions so many measurements can be related to the same unknowns (the $p_{i,j}$). As a consequence, the rank of the system can be enough increased and the system can be solved.

2 Related Work

Problem (1) is in fact an inverse problem. The matrix A is not square so it cannot be inverted. Actually, this system is under-constrained : there are less equations than unknowns. So we are able to find solutions of the system (1) but we can not know which one is the actual one.

Since link counts alone do not allow to solve for the actual traffic matrix, additional information is required. Previous works on traffic matrix estimation mainly differ in the way they introduce this additional information. A short overview of these previous works is provided in the following. Interesting reader may refer to [4] where a full reviewed of the oldest methods is done and to [9] for newer methods description.

2.1 Use of link count covariance and temporal prior information

A first generation of methods has tried to use information about link load covariances (most of the time named the link count covariances) which can be related to the traffics and so increase the rank of the system. In fact, as link counts and traffics are related by the linear application A, we can write the covariance of the link counts as $Cov(y_i, y_j) = \sum_{k=1}^{c} B_{ij,k} x_k$ where $B_{ij,k} = A_{ik} A_k j$. In mathematical terms

$$\begin{pmatrix} Y \\ Z \end{pmatrix} = \begin{pmatrix} A \\ B \end{pmatrix} X \tag{2}$$

Where Z contains the covariances of the link counts. Vardi, in [12], was the first to use this by assuming Poissonian model for the traffics. He also proves that the system (2) is identifiable (solved by an EM-algorithm). But his results are quite poor as [4] proves it because Poissonian assumption is not verified in real Networks. Then Cao et Al. consider a Gaussian model in [1] for the traffics with respect to the power law : a relation between mean and variance of the traffics given by

$$\Sigma = \Phi diag \lambda^c \tag{3}$$

Where Σ is the covariance matrix of the traffics (assumed to be diagonal) and λ is the vector of traffic's mean. Cao et al. use a modified EM-algorithm to solve their problem. Although this algorithm is really heavy in term of computation, it gives some good results in term of performance. Nevertheless Medina et al. prove in [4] that this Gaussian assumption is not always verified in real networks. The values of the model may change for different hours of measures on it is completely false for certain networks. However, Medinal et al. claim in that the power law is verified on real Networks.
More recently, Vaton et al. show in [3] that good results can be obtained without any prior assumption except the one about power law. They propose a fast algorithm to obtain estimation on the means of the traffics. Their results are worse than the EM-algorithm but the gain regarding computing time is consequent. What's more, they propose in [7] an analytic formula to compute the Cramér-Rao lower bound. This is useful to compare the results of estimating algorithm to EM-algorithm because the latter reach this bound.

2.2 Use of spatial or temporal prior information only

As these methods based on prior information about traffic model are limited, some authors have considered a second generation of methods for traffic matrix estimation. They use prior information too but this time, this information is either spatial or temporal. The gravity model for example [13] assumes that the matrix follows a gravity rule (spatial relations).The traffic between two nodes i and j is proportional to the incoming traffics at node i and to the outgoing traffics at node j.

$$x_{i,j} \propto x_{i,*} \times x_{*,j} \qquad x_{i,*} = \sum_{j \neq i} x_{i,j} \tag{4}$$

A regularization approach can be adopted to find the solution of S the closest to the gravity model. The distance can be the usual L_2 norm to obtain the least-square solution as in [13] but it can also be the Kullback-Leibler distance as in [10]. In other terms, if the infinite space S of solutions of (1) and the gravity solution X_G have been generated, then we have to solve

$$\min_{X \in S} ||X - X_G|| \tag{5}$$

Another way to add prior information is to make many measures while changing the routing scheme for each measure [10] (temporal relations). So if τ measures are done with a different routing scheme A^t each time, the system (1) becomes

$$\begin{pmatrix} Y^1 \\ Y^2 \\ \vdots \\ Y^\tau \end{pmatrix} = \begin{pmatrix} A^1 X \\ A^2 X \\ \vdots \\ A^\tau X \end{pmatrix} \tag{6}$$

In [5] authors propose an heuristic to compute the minimum number of weight changes in order to have a full rank matrix in (6) and so a solvable system. But, even if the minimum number of routing changes can be found, changing the routing scheme many times will never be accepted by network providers.
We can wonder what happens on performances if the prior information is false. Without any surprise, in this case, the estimate is really bad. Make an assumption on the traffic model is very hard and the gravity model for example isn't always verified in real networks. That is why all the solutions using a prior information can't give a good approximation in real networks unless a realistic spatial model for traffics is found.

2.3 Use of spatial and temporal posterior information

To cope with the latter comment, a third generation of methods, well explained and tested in [10], are looking at the real traffics behaviour. In fact, a model is built from real measures (obtained by Netflow) and then this model is used to predict future matrix.

[11] proposes for example a kalman filter calibrated from real Netflow measures made during 24 hours to estimate traffics. In the paper, authors propose to consider traffic matrix estimation as a kalman filter prediction. To do that, they write the following dynamic linear equations for X and Y

$$\begin{cases} X_{t+1} &= C_t X_t + W_t \\ Y_t &= A_t X_t + V_t \end{cases} \quad (7)$$

where the state noise W_t and the measurement-noise V_t are supposed to be uncorrelated and zero-mean Gaussian white noise processes with respective covariance matrices Q_t and R_t. Soule et al. calibrate this model thanks to Netflow measures by assuming A,C,Q and R are constant. They use a method of Expectation maximisation (EM) based on the measured data to solve the parameter estimation problem. The next steps to estimate traffics are the two well known processes :

- Prediction Step

$$\begin{cases} \widehat{X}_{t+1|t} &= C\widehat{X}_{t|t} \\ P_{t+1|t} &= CP_{t|t}C^T + Q \end{cases} \quad (8)$$

- Update Step

$$\begin{cases} \widehat{X}_{t+1|t+1} &= \widehat{X}_{t+1|t} + K_{t+1}[Y_{t+1} - A\widehat{X}_{t+1|t}] \\ P_{t+1|t+1} &= (I - K_{t+1}A)P_{t+1|t}(I - K_{t+1}A)^T \\ & \quad +K_{t+1}RK_{t+1}^T \\ K_{t+1} &= P_{t+1|t}A^T(AP_{t+1|t}A^T + R)^{-1} \end{cases} \quad (9)$$

Soule et al. don't explain how they are sure of the observability of the system but it seems that the presence of matrix C_t in (7) makes the system observable.

Another particularity of the traffics behaviour is used in the Fanout method. Papagiannaki in [6] shows that even the traffics are not constant, the proportion in the direction of j of all the traffics arriving in node i are merely constant during one hour and what's more, these proportions named fanout are nearly the same between two days for the same hour. So the idea is to measure this fanout for each hour during one day using Netflow measures and to use them after to predict traffics for the next days. We will explain in detail this spatial prior in the next section as we use it in our problem statement.

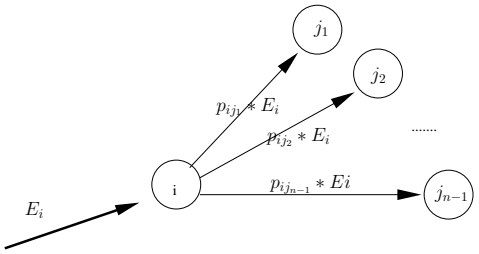


Figure 1. Description of fanout

All these methods using more expensive measures to calibrate a model have two hard points

- 24 hours of Netflow measures appears to be too heavy for network providers. These measures will slow the network down for one entire day. However some providers may agree with this utilisation if beneficial contribution is proved.
- What's more, there is not only one day of measures since the calibration becomes quickly out of date. That's why these methods contain a detection of the error. As soon as the calibration appears to be out of date, another calibration phase is launched. So the impact of these methods on network is consequent.

To cope with these issues, we propose here another estimation based on link count only. We will suppose our traffics have the pattern of the fanout and we show that under this assumption, our results are better than the oldest deterministic methods and pretty much as good as the results of the last generation of methods.

3 The Fanout Approach

We decide to use the fanout model which seems to be realistic as we explained in the section 2. As we can see on Figure 1, the idea is to consider the traffic x_{ij} as the incoming traffic at node i : $E_i = \sum_{j \neq i} x_{ij}$ multiplied by the proportion p_{ij}. The c-vector P containing the p_{ij} is named fanout.

The idea hidden behind the fanout is to link the traffic to something measurable by SNMP, the E_i and to another thing we have to estimate (the fanout). So we add some information by using the E_i values but it is not enough. Another assumption or properties need to be used. In [6] some properties about the fanout can be read : The fanouts tend to show very limited variations around their mean value for the same hour across days. So

- The value of the p_{ij} are quite constant on a period of one hour

- The fanout P are nearly the same from one day to another

Moreover, some values for the variation are given in the paper. For the same hour across day, the fanouts variation is included between 0.1 and 0.5. Soule et al. in [10] propose to measure these fanouts for one day using Netflow in order to predict future traffic from them and from SNMP measures (to have E_i). We propose to skip using Netflow and to use many measures for one hour to estimate in the same time the fanout and so the traffics. We assume too that the routing matrix is constant.

Let's come back on the notations used to pose the problem. The unknowns are contained in the c-vector $\mathbf{P}$. We will use two different notation for $\mathbf{P}$: we may name them p_i where i is an Origin-Destination pair (OD) so $1 \leq i \leq c$ or we may note them $p_{i,j}$ with i which is the origin node and j the destination node so $(i,j) \in [|1,N|]^2$ with $i \neq j$.

We still have the routing matrix A and the link count Y. As we will do many measures, the link count for the measure t will be denote by Y^t. And the c-vector E^t will represent the traffics getting in the network at each edge for measure t. It contains $N-1$ times the traffic getting in the network by edge node 1 (E_1^t) then $N-1$ times E_2^t etc ....

$$E_i^t = \underbrace{(E_i^t, E_i^t, \ldots, E_i^t)}_{N-1} \qquad E^t = \begin{pmatrix} (E_1^t)^T \\ (E_2^t)^T \\ \vdots \\ (E_N^t)^T \end{pmatrix} \quad (10)$$

If the routing equation from (1) is written with these new notations, it gives for the measure t

$$Y^t = APE^t \quad (11)$$

As this system is still under-constrained, many measures are needed to increase the rank to c. So τ measures will be done and the problem is now to find P which solves the system

$$\begin{pmatrix} Y^1 \\ Y^2 \\ \vdots \\ Y^\tau \end{pmatrix} = \begin{pmatrix} APE^1 \\ APE^2 \\ \vdots \\ APE^\tau \end{pmatrix} \quad (12)$$

If τ is chosen so that (12) has a full rank, then a singular solution P exists. But this is the ideal case. In reality, the fanout will change a bit between each measure. So a minimisation problem has to be solved

$$\min_P \quad \{ \quad \sum_{t=1}^{\tau} ||Y^t - APE^t||^2 \quad \} \quad (13)$$

Under the constraints :

$$\forall i = 1..N \quad , \quad \sum_{j \neq i} p_{i,j} = 1$$
$$\forall i = 1..c \quad , \quad p_i \geq 0$$

We want to estimate the fanout so that the sum of the error for each snapshot is minimised. As the fanout must stay quite constant, we think a mean value can be found for them. We hope that this estimation can be used to well estimate the OD flows. In fact, this method has to be used to have the mean of the flows for a while. If we really want to know how are the traffics for one slot of 5 minutes, a proportional fitting algorithm may be used in order to respect (11) because the minimum reached won't respect this equation. We won't come back on this point because the following target is focused : to obtain one fanout for each set of measures (most of the time during one hour). And to do that, the assumption is to consider the fanout constant during all the snapshot of this set of measures.

It is easy to show that the space of solutions P is convex. What's more, the minimisation function is convex too. So the problem (13) is in fact the minimisation of a convex quadratic function on a convex space and many methods exist to solve such problem of minimisation. However, most of them don't cope with constraints. So more specific algorithm have to be used. We tried to use a projected gradient minimisation and we have had good results for small networks but the convergence was too long, sometimes even never reached for bigger networks. We tried to use second order descent method but the results were still bad. In fact, we tried many descent method, even some we have created. As we studied a lot the particularity of our system, we tried to project in some descent direction which was more attractive than others (such as the one of the projected gradient). In fact, it is the sum constraint and the positivity constraint together which are compelling sometimes some solutions close to 0 to decrease. So the step of the descent method become too short and the convergence is not reached before a long long time. To go against this, we propose a local search method which is very quick and efficient. It will be described in the next section.

4 Local search to solve the Minimisation Problem

The local search heuristic is an iterative method which brings the current solution from an initialisation point to a local optimum of the problem. The main feature of this heuristic is the choice of the solution P^{k+1} from P^k. In fact, a neighbourhood V^k of P^k must be defined by a transformation T such as $V^k = T(P^k)$. Then P^{k+1} will be

chosen among the elements of V^k with respect to certain rules. The most important thing is that T must keep all the new available solutions as feasible solutions. For all k and for all $P \in V^k$, P must respect the constraints (14). The algorithm will start with an initial solution P^0 chosen to be feasible. Then each new solution P^k will be feasible too thanks to T properties. So the transformation T and the choice's rules are two specific parts of a local search algorithm. We will detail them in the next parts of the paper.

4.1 Neighbourhood

We decide to use a local search heuristic in order to avoid the problem caused by the constraints. So a transformation which can move some bigger or smaller part of the traffic from one OD to an other has to be found while avoiding to touch the other OD. We introduce a new parameter δ_k which depends on the iteration k. This parameter will represent the amount of traffic to move. As it can be seen on the Figure 2, on the left part of the figure is the solution P^k and on the right part is a solution owning to V^k. The only differences between the two are the values of p_{uv} and p_{uw}. p_{uv}^{k+1} has been decreased by δ_k compare to p_{uv}^k and p_{uw}^{k+1} has been increased by δ_k compare to p_{uw}^k. In term of OD, $E_u\delta_k$ units of traffic are moved from OD uv to OD uw. If this is written in a mathematical form, then

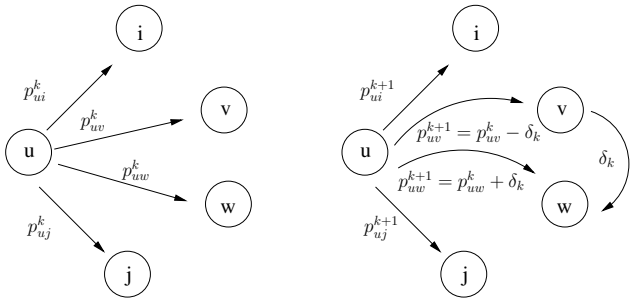


Figure 2. Neighbourhood Definition

$$P \in V^k \text{ iff } \exists!\ (u,v,w) \in \{1,\dots,c\}^3 \text{ such as } \forall ij = 1..c$$

$$p_{ij} = \begin{cases} p_{ij}^k & \text{if } ij \neq uv \text{ and } ij \neq uw \\ p_{ij}^k + \delta_k & \text{if } ij = uw \\ p_{ij}^k - \delta_k & \text{if } ij = uv \end{cases}$$

This neighbourhood has $N(N-1)(N-2)$ elements so it is quite huge. As no particularity between two elements of V^k can be found, it is not possible to find a realistic and efficient way to explore only a part of it (except a probabilistic way as in the simulated annealing method). So we need to explore all of the neighbours therefore we have to compute the new value of the criteria for each element of V^k. Fortunately, we have an analytic formula to compute the variation of cost between P^k and an element of V^k. Another important thing is the presence of the parameter δ_k. Thanks to this parameter, the part of traffic moved to one OD pair to another can be adjusted by this parameter. This will be useful when no neighbours appears attractive, in this case, another neighbourhood will be built with δ_{k+1} smaller. We will come back on this point later when explaining our algorithm.

4.2 Evaluation of the neighbours

The objective function associated to P is denoted by Γ

$$\Gamma(P) = \sum_{t=1}^{\tau} ||Y^t - APE^t||^2 \tag{14}$$

The first idea to evaluate the cost of P (element of V^k) is to compute $\Gamma(P)$ from nothing. But this computation is quite long if the number of nodes N is high. So doing it $card(V^k)$ times will be too long. What's more, it is known from the neighbourhood definition that there are not so many changes in the loads compare to the current solution X^k. That's the reason why an analytic formula has to be found in order to compute the variation of the objective function between the current solution P^k and an element of V^k.
To do this, we suppose knowing the cost of a solution P and we look after the variation of cost Δ_{uv} obtained after adding $\delta_{u,v} = +/-\delta$ to parameter p_{uv} associated to OD pair k. Another notation is needed to well explain this, the load generated by the solution P is written by $\overline{Y^t} = APE^t$. Under these assumption

$$\Delta\Gamma_{u,v} = \sum_{t=1}^{T} \sum_{l \in L_{uv}} \frac{\Delta y_{k,l}^t}{{y_l^t}^2} \left[2(\overline{y_l^t} - y_l^t) + \Delta y_{k,l}^t\right] \tag{15}$$

$$\text{With , } \forall l \in L_{uv} \quad \Delta y_{k,l}^t = \delta_{uv}\, a_{l,k}\, E_u^t$$

where L_{uv} contains the links l used by the OD pair k (such as $a_{l,k} \neq 0$). The simplification is in fact to compute the new link load only for the links where it has changed. By avoiding to repeat the same computation for the unchanged links, a lot of time is won (think of the size of V^k).

4.3 Initialisation

As we said it before, a local search need an initial solution which is feasible with respect to the constraints of the problem. To do this, it is possible to use initial solution already used in the literature like gravity solution but we

Algorithm 1 Thousand Leaves

1: $\forall i = 1..c \quad p_{ij} = 0$
2: $\forall i = 1..N \quad q(i) = 0$
3: $\Delta_{min} = 0$
4: $\Gamma = 0$
5: **while** $\sum_i q(i) < \frac{N}{Q}$ {Until all quantum are placed} **do**
6: **for** $i = 1$ to N {For each Origin} **do**
7: **if** $q(i) < \frac{1}{Q}$ **then**
8: **for** $j = 1$ to N {For each Destination} **do**
9: **if** $i = j$ **then**
10: continue
11: **else**
12: Evaluate $\Delta\Gamma_{ij}$ with $\delta_{ij} = +Q$
13: **if** $\Delta\Gamma_{ij} < \delta_{min}$ **then**
14: $i^* = i$
15: $j^* = j$
16: $\delta_{min} = \Delta\Gamma_{ij}$
17: **end if**
18: **end if**
19: **end for**
20: **end if**
21: **end for**
22: $p_{i^*j^*} += Q$ {Add Q where it increases at least the cost}
23: $q(i^*) += 1$ {Count how many quantum have been added on i}
24: Compute $\Gamma(P)$ {Update the cost}
25: **end while**

prefer to use an initial solution close to our neighbourhood definition.

To remember, the constraints which have to be respected are the positivity of each p_{ij} and the fact that for each edge node i, $\sum_{j \neq i} p_{ij} = 1$. The first constraint is easy to respect so the only problem deals in the second one. Sure a proportional solution where $p_{ij} = \frac{1}{N-1}$ may be used but our experiment showed that this solution is not good. So we decide to take a little number Q named quantum such as $Q << 1$ and we use an algorithm to place them in order to make our initial solution. To place a quantum on p_{ij} means that p_{ij} will be increased by Q. Therefore we need to start with $p_{ij} = 0$ for all ij and the necessary $\frac{N}{Q}$ quantums are placed by using the algorithm 1 named *thousand-leaves*. The idea used in this algorithm is to place each quantum on the OD pair such as the increase of cost is the minimum. To do that, we make a loop on the origin i (line 6) and a loop on the destination j (line 8) in order to evaluate the cost variation if p_{ij} is increased by Q. To make the evaluation (line 12) the analytic formula (15) is used with $\delta_{ij} = +Q$. The algorithm stops when $\frac{N}{Q}$ quantums have been placed. The solution obtained by this algorithm is feasible because the algorithm stop placing any quantum on origin i if $q(i) = \frac{1}{Q}$ (line 7), and $q(i)$ is increased by one each time a quantum is placed on an OD pair whose origin is i. So, if the number of quantum placed is denoted by n_{ij} on the OD pair ij then $\sum_{j \neq i} p_{ij} = Q \sum_{j \neq i} n_{ij} = Q q(i) = 1$. This algorithm allows us to obtain a feasible solution and the way to build it is close to the neighbourhood structure thanks to the leaves.

4.4 Algorithm Generalised FanOut : GFO

The local search we have implemented and named GFO is as simple as fast and efficient. The pseudo code is written in Algorithm 2. During the initialisation phase (line 1-4),the algorithm thousand-leaves is used with $Q = 2e-4$ so we know that by taking $\delta_0 \geq Q$ there will be no better solution in V_0. That's why the algorithm start with $\delta_0 = \frac{Q}{2}$. Then (line 7-14) the neighbourhood is explored and the boolean *better* allows us to know at the end of the exploration if a better solution P_{min} is found. If so, δ is left unchanged and $P^{k+1} = P_{min}$, if not, δ is divided by two and P^k is left unchanged. The function *needToStop()* returns true if and only if one of the following conditions holds :

1 δ_k becomes too tiny ($< 1e^{-6}/(N-1)$)
2 The maximum relative error on the loads is under 1%
3 The criteria doesn't evolve anymore

Algorithm 2 GFO

1: P^0 = thousand-leaves() {Initialisation}
2: $\Gamma^* = \Gamma(P^0)$
3: $k = 0$
4: $\delta_0 = \frac{Q}{2}$
5: **while** not needToStop() **do**
6: boolean better = false
7: **for all** $P \in V_k$ {For all neighbours of P^k} **do**
8: Evaluate $\Gamma(P)$
9: **if** $\Gamma(P) < \Gamma^*$ **then**
10: $\Gamma^* = \Gamma(P)$
11: $P_{min} = P$
12: better = true
13: **end if**
14: **end for**
15: **if** better **then**
16: $P^{k+1} = P_{min}$
17: $\delta_{k+1} = \delta_k$
18: **else**
19: $P^{k+1} = P^k$
20: $\delta_{k+1} = \frac{\delta_k}{2}$
21: **end if**
22: **end while**

5 Tests and Results

We didn't do our test on real data. In fact, some traffic matrices have been created with respect to real properties, then we have propagated these flows on an hypothetical network using NEST. Thanks to this tool, it is possible to create a network with routers and links, to compute the routing matrix, to propagate the flows, to obtain the link load after propagation. So our test are done as following :

1. Create a network and get A from NEST ([8])
2. Create traffic matrices X^t
3. Propagate this matrices and get the link load Y^t
4. Run the algorithm from A and Y^t
5. Compare the solution obtained to the original X^t

We will first explain how we build traffic matrices, then we'll give the specifications of the network of test, finally we'll give the numeric results.

5.1 Generation of original traffic matrices

To generate as many matrices as measures, an initial matrix X^0 is first generated then this matrix is made evolving for each instant of measure. It can be read in most of the paper that only 30% of the flows are representing 80% of the total load in usual traffic matrices. Moreover, we know there are most of the time bigger flows, middle flows and little ones. So we decide to generate our traffic matrix according to a multi-modal Gaussian distribution. We make a distribution which is a weighted sum of three Gaussian. If the Gaussian distribution of mean λ and variance σ is written by $G(\lambda, \sigma)$, then the probability distribution for x_{ij} is

$$P = \sum_{k=1}^{3} p_k G(\lambda_k, \sigma_k)$$

$$\begin{pmatrix} p_1 = 0.3 & \lambda_1 = 50000 & \sigma_1 = \frac{\lambda_1}{10} \\ p_2 = 0.2 & \lambda_2 = 10000 & \sigma_2 = \frac{\lambda_2}{10} \\ p_3 = 0.5 & \lambda_3 = 500 & \sigma_3 = \frac{\lambda_3}{10} \end{pmatrix}$$

From this initial matrix X^0, initial values for $E_i^0 = \sum_{j \neq i} x_{ij}^0$ and initial values for P^0 by $p_{ij}^0 = \frac{E_i^0}{x_{ij}^0}$ are computed. These values will be the means for these parameters. So in order to create a different matrix for each measure t, the following is done

1. Evolution of the traffics getting in the network by i

$$\forall i = 1..N \qquad E_i^t = E_i^0 + G(0, \frac{E_i^0}{10})$$

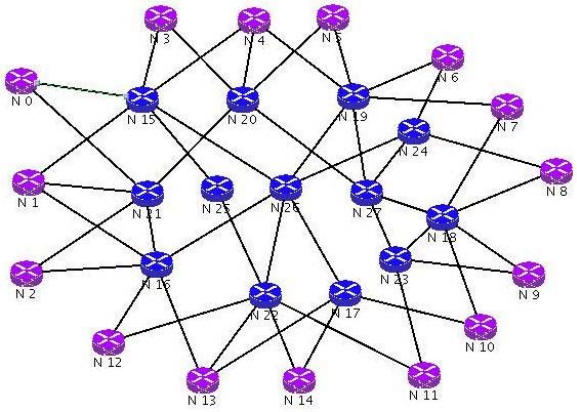
2. Evolution of the fanout

$$\forall ij = 1..c \qquad p_{ij}^t = p_{ij}^0 + G(0, p_{ij}^0 \times pVar)$$

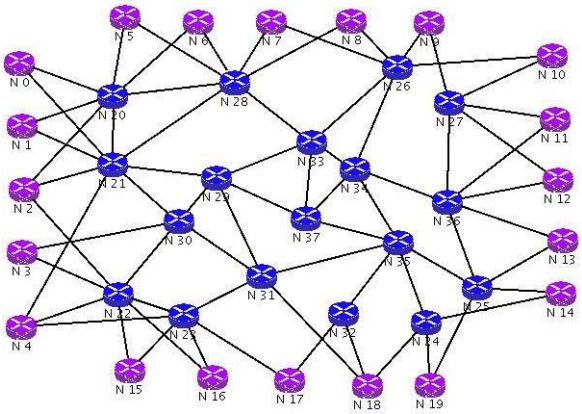
3. Traffic Matrix for instant t

$$\forall ij = 1..c \qquad x_{ij}^t = p_{ij}^t \times E_i^t$$

After this generation step, we eventually have our τ matrices and we propagate them in the network according to the routing matrix. As we already said it before, this last step necessary to have the link load is done with NEST.
So we are able to generate many measures with respect to some properties. The parameter $pVar$ allows us to quantify the variation of the fanout between each measure. When we pose the minimisation problem in (13), it was under the assumption that the fanout are varying a few during measures for one hour period. So if $pVar$ is too high, our assumption is not verified anymore. We will test the algorithm for $pVar$ values around some percents, most of the time 1% and sometimes more.



(a) *net1*



(b) *net2*

Figure 3. Test networks

5.2 Test Networks

We propose to show results obtained on different networks. The first one *net1* is a 15-edges network showed in the top of the figure 3 and on the bottom of the same figure we put the second one *net2* : a 20-edges network. *net1* has 28 nodes whereof 15 nodes are Origin/Destination so 205 pair OD, it contains also 98 links. *net2* has a total of 38 nodes whereof 20 nodes are edges so 380 pair OD and *net2* contains 142 links. The routing matrix used is not the usual one used in the literature since the OSPF balanced sharing along paths of the same length is used. So the values in the routing matrix are not only zero or one. It is possible to do that easily thanks to the routing part of NEST which computes the routing matrix in few seconds from OSPF metrics on the links.

5.3 Numerical Results

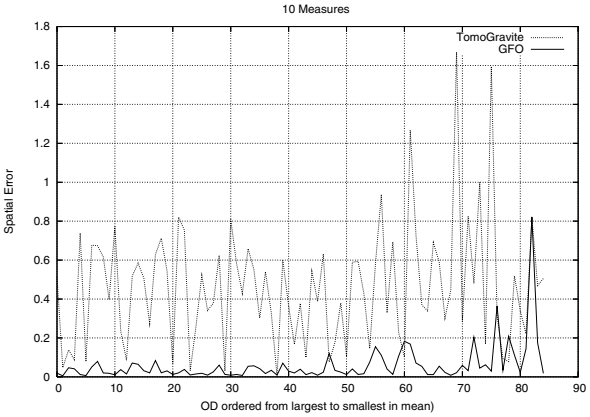


Figure 4. Spatial Error on *net1*

In order to compare our algorithm with results we have seen in the literature, we evaluate two often used criteria :

- The spatial error for each OD pair uv : it summarises the error of OD flow uv over its lifetime.

$$errS_{uv} = \frac{\sqrt{\sum_{t=1}^{\tau} \left(X_{uv}^t - P_{uv}E_{uv}^t\right)^2}}{\sqrt{\sum_{t=1}^{\tau} {X_{uv}^t}^2}} \qquad (16)$$

- The temporal error for each measure t : it summarises the error on all OD flows for an instant t.

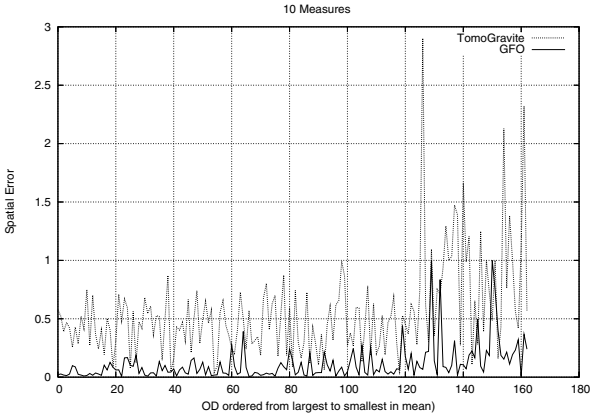


Figure 5. Spatial Error on *net2*

$$errT_t = \frac{\sqrt{\sum_{ij=1}^{c} \left(X_{ij}^t - P_{ij}E_{ij}^t\right)^2}}{\sqrt{\sum_{ij=1}^{c} {X_{ij}^t}^2}} \qquad (17)$$

Another metric is computed to evaluate our results : the mean of relative deviation between the traffics. In fact, we compute the relative deviation between the generated traffics and the evaluated one and we take the mean over all OD pairs and time. This error is denoted by $errA$ and

$$errA = \sum_{t=1}^{T} \frac{1}{\tau} \sum_{uv=1}^{c} \frac{1}{c} \frac{\left(X_{uv}^t - P_{uv}E_u^t\right)}{X_{uv}^t} \qquad (18)$$

We need to precise here that the values X_{uv}^t are in fact the real traffics, and the values $P_{uv}E_u^t$ are the traffic for OD pairs uv which are estimated by our algorithm. The last point to precise is the number of flows whose results are presented. It is known that estimate the smallest flows is very hard. As their impact on the entire matrix is very poor, it is not so dramatic to bad estimate these flows. So all the results given later are only for flows, ordered by decreasing mean, which count for 95% of the total load.

We didn't lmimplemented the newest methods (PCA, Kalman or Fanout) because we didn't have access to real consequent data. We tried to implement methods such as EM-algorithm but the results were worse than the one announced certainly because of our implementation. So to compare our algorithm to these methods, we need to have a look to the papers which are dealing with them. Good references in term of results are [10] where they have tested tomogravity, weight change, fanout, PCA and Kalman methods, [9] where authors have tested many methods based on

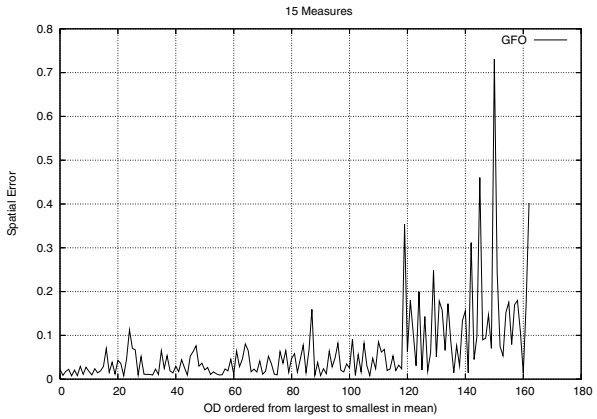


Figure 6. Spatial Error on *net2*

EM-algorithm and all its improved variant.
Nevertheless, we propose to compare our algorithm to the tomogravity method explained in section 2.2. Even we know the results of this methods will be poor because our matrices are not following a gravity scheme.

5.3.1 Results for $pVar = 1\%$

In the Figure 4 and 5 we can see the spatial error respectively for *net1* and *net2* for $\tau = 10$ measures. The errors obtained are very good if we compare them to the literature. We don't exceed 20% except for the last traffics for *net1*. For the second network, results are a bit worse but this is due to the number of measures. In term of average error, we found that $errA = 5.88\%$ for *net1* and $errA = 11.4\%$ for *net2*

If we look at the Figure 6 where we only print the spatial error of GFO for $\tau = 15$ measures we see that the errors are under 10% most of the time. We reach 20%-30% for the last OD flows. By increasing the number of measures, we found $errA = 6.3\%$ this time (compare to 11.4% with 10 measures). So the number of measures is important for our estimation process. The question to find the minimum of measures we need to have a good solution is an important perspective for traffic matrix estimation. We will come back on this point later.

We didn't print the temporal errors for *net1* and *net2* since we have a constant error rate (around 0.6 for the tomogravity and around 0.1 for GFO) for each measure but this leads us to two comments.

First, we have to compare this to the third generation methods which have temporal error completely different. Their temporal error increases constantly after a calibration phase until the calibration is considered out of date. So the estimations of these methods are better after the calibration than later.
Second, we explained in section 3 we want to estimate one fanout for one period. So the fact we have the same error rates for all measures is a good point. It means that by minimising the sum of the error on the load (cf equation 13) with this algorithm, we have approximatively the same error rate for each measure. So we are sure our estimation will be good for all the periods.

5.3.2 Results for $pVar > 1\%$

We try to increase $pVar$ to see how our approach and algorithm will behave for higher values of $pVar$. As we made the assumption that the fanout are varying a few, we limit $pVar$ at 10%. For this results, we increase the number of measures : $\tau = 15$ in order to show something we will explain a bit further.
The Figure 7 and 8 show the results obtained. The first comment we can do is the fact that the error increases with the value of $pVar$. Then, we can see on these figures that with a value of $pVar$ under 5%, the results are quite acceptable. Over this value, the results are really to bad but our assumption is a **low** variation of fanout...

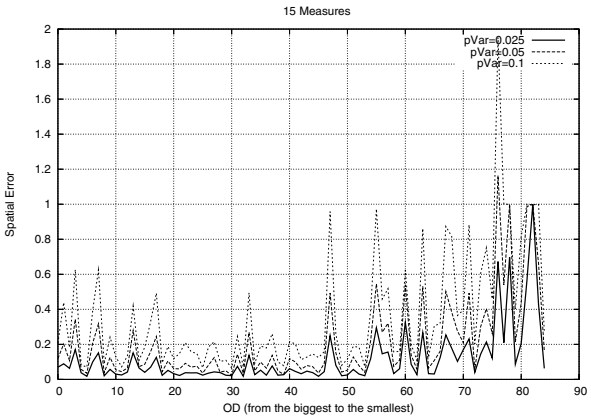


Figure 7. Spatial Error on *net1* **for different** $pVar$

What we wanted to show by increasing the number of measures to 15 is on the figure 7. On *net1*, we had really good results for 10 measures. We can think that the fact to increase the number of measures will leads us to increase our performance. However, on this Figure, we can see that some big traffics are not well estimated and this is the worst fault for an algorithm of traffic matrix estimation. So we need to find the best number of measures in order to have the best estimation.

6 Conclusion

Our contribution in this paper has a double aspect. First, we propose a new approach for traffic matrix estimation

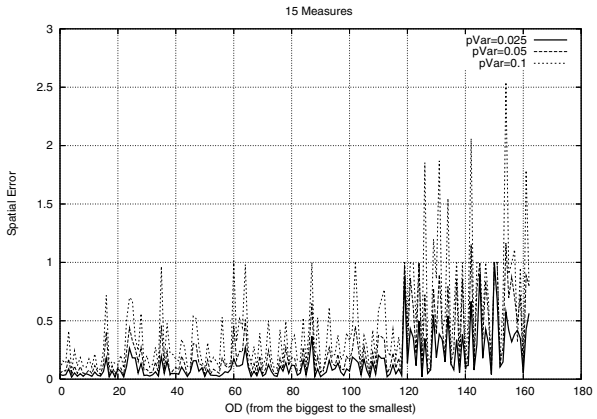


Figure 8. Spatial Error on *net2* for different $pVar$

which avoids to use Netflow measures but permits to obtain results pretty much as good as methods which are using them. By comparing our method to the referent tomogravity, we show we are really better. Sure we make the assumption about the low variability of the fanout but Papagiannaki et al. have shown in [6] that this assumption is right. So this approach seems the most efficient in its category.
Then we propose an algorithm really simple to solve our problem. We know that exact methods exist but the advantages of the GFO algorithm is its simplicity. This heuristic permits us to obtain a global minimum of the problem in really short computing time.

There are numerous perspectives on the subject and on our approach. We first want to try this method on real data to see if our assumption and resolution method are so efficient on real data than on simulated one. Then, we think the issue to find the best number of measures is really interesting and must be study. To do that we already have the idea to make a progressive algorithm which estimates the solution for one measure, then for two, etc ... and stops when the new solution isn't better. At last, analysing why some traffics (even bigger ones) are badly estimated when $pVar$ increases compare to other ones which still are well estimated appears to be another interesting point to develop.

References

[1] J. Cao, D. Davis, S. Wiel, and B. Yu. Time-varying network tomography : Router link data, 2000.

[2] Cisco. Cisco ios netflow http://www.cisco.com/en/us/products/ps6601/products_ios_protocol_group_home.html.

[3] I.Juva, S.Vaton, and J.Virtamo. Quick traffic matrix estimation based on link count covariances. *2006 IEEE International Conference on Communications (ICC 2006),Istanbul*, 2006.

[4] A. Medina, N. Taft, S. Battacharya, C. Diot, and K. Salamatian. Traffic matrix estimation: Existing techniques compared and new directions, 2002.

[5] A. Nucci, R. Cruz, N. Taft, and C. Diot. Design of IGP link weight changes for estimation of traffic matrices. In *IEEE Infocom*, Hong Kong, March 2004.

[6] K. Papagiannaki, N. Taft, and A. Lakhina. A distributed approach to measure IP traffic matrices. In *Internet Measurement Conference*, pages 161–174, 2004.

[7] P.Bermolen, S.Vaton, and I.Juva. Search for optimality in traffic matrix estimation: a rational approcah by cramer-rao lower bounds. *2nd EuroNGI Conference on Next Generation Internet Design and Engineering, Valencia, Spain*, 3-5 april 2006.

[8] QoSDesign. Nest : Network engineering and simulation tool http://www.qosdesign.com.

[9] V. Sandrine, B. J.s, and G. Anni. Advanced methods for the estimation of the origin destination traffic matrix. *Advanced Methods for the Estimation of the Origin Destination traffic matrix. Performance Evaluation and Planning Methods for the Next Generation Internet (25 th du GERARD)*, XVI, 2005.

[10] A. Soule, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, and C. Diot. Traffic matrices: balancing measurements, inference and modeling. *SIGMETRICS Perform. Eval. Rev.*, 33(1):362–373, 2005.

[11] A. Soule, K. Salamatian, A. Nucci, and N. Taft. Traffic matrix tracking using kalman filters. *SIGMETRICS Perform. Eval. Rev.*, 33(3):24–31, 2005.

[12] Y. Vardi. Comment: Bayesian inference on network traffic using link count data. *Journal of the American Statistical Association*, 93(442):573–??, June 1998.

[13] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads, 2003.