THE SINGLE CLASS TRAFFIC ASSIGNMENT TOOLBOX OF SCILAB : CIUDADSIM

P. LOTITO, E. MANCINELLI, J.P. QUADRAT AND L. WYNTER

Contents

1. Introduction to CiudadSim

The Traffic Network toolbox of Scilab called *CiudadSim* is intended to study the assignment problem in traffic networks that is to compute the traffic on the links of a Network knowing the transportation demands for some couple of origine and destination of the given network. For that CiudadSim adds dedicated functions and data structures to Scilab. It use Scigraph (a toolbox of Scilab) to visualize and edit the network.

CiudadSim is a contribution of Scilab which is free and available at www-rocq.inria.fr/scilab/CiudadSim. It needs :

- *Scilab* available at
  www-rocq.inria.fr/scilab/,
- *Scigraph* a Scilab toolbox for visualizing graph available at
  ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/SCIGRAPH/
- *MaxPlus* a Scilab toolbox for max plus arithmetic available at
  ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/MAXPLUS/

CiudadSim can be decomposed schematically in two parts.

- A database called `%net` which contains all the given data – geographic definition of the network, – travel demands (*commodities*), – travelling time as functions of congestion (link performance functions). It contains also all the computed data – the aggregated and diasaggregated (with respect to the commodities) flows on the links, – the times spent on each link at equilibrium. The function `TrafficExample` builds the database for some examples. The function `ShowNet` visualizes the net in an editable window. It is possible to edit the database using the menus of this visualizing window.
- A Scilab function `TraffigAssig` able to compute traffic assignments (Wardrop equilibrium, logit and probit equilibrium) using various algorithms. The results are written in some fields of `%net`. The assignment obtained can be visualized using `ShowNet`. The widths of the links indicate their degrees of congestion.

2. The database `%net`

A traffic network has three well distinguished parts. The nodes, the links and the demands. The data set for each of those parts will be as well different. For example for the nodes we will only need there coordinates but for the links we will need together with their head and tail nodes, the corresponding link performance functions. We will consider the O-D pairs together with the corresponding travelling demands as arcs directed from the origin to the destination with the demand as associated weight.

All this data are collected in a Scilab typed list called `%net` which play the role of a geographic database. It is defined by

```
%net=tlist(['net','gp','nodes','links','demands'],gp,nodes,links,
demands)
```

where:

- `gp` : is a tlist containing the general properties of the net,
- `nodes` : is a tlist containing the nodes parameters,
- `links` : is a tlist containing the links parameters,
- `demands` : is a tlist containing the demands parameters.

The exhaustive list of the fields is given in the manual section below at the Netlist subsection. Let us make some remarks on some non trivial fields.

The link performance function of each link is a function of the flow f on the link described by two fields :

- `%net.gp.lpf_model` which is a number corresponding to a model of congestion among :
  (1) $t(f) = t_0 + \frac{m}{c} f + m \max(0, f - c)^{\beta}$
  (2) $t(f) = t_0 \exp\left(\frac{f}{c} - 1\right)$

(3) $t(f) = t_0\, 2^{\left(\frac{f}{c}-1\right)}$
(4) $t(f) = t_0 \left(1 + 0.15 \times \left(\frac{f}{c}\right)^m\right)$
(5) $t(f) = t_0 + \log\left(\frac{c}{c-f}\right)$
(6) $t(f) = \beta - c\frac{t_0-\beta}{f-c}$
(7) $t(f) = t_0 + c\, f + m\, f^\beta$

- `%net.links.lpf_data` which contains for each links the values of the four parameters t_0, c, m, β appearing in the definitions of the link performance function choosen. Where – t_0 is the free-flow travel time, which is a measure of the travel time at zero flow, – c is the practical capacity of the link, which is a measure of the quantity from which the travel time will increase very rapidly.

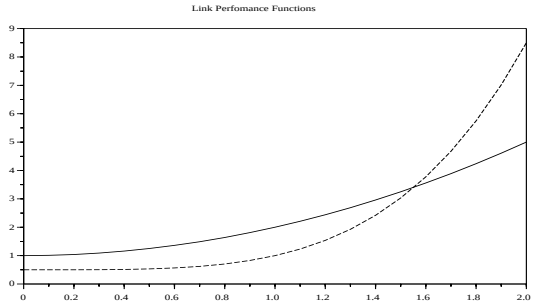

FIGURE 1. Example of Link Performane function

It is possible to plot the link performance funtion using `dlpf` (draw link performance functions) for example :

```
lpp=[1 0 1 2; 0.5 0 0.5 4]';
dlpf(0,0.1,2,lpp,6);
```

The method to compute the assignment is given in the field `algorithm` of `%net.gp` the default one is the DSD described below. The main results of the assignment are stored in the fields `%net.links.flow` and `%net.links.time`.

The function `TrafficExample` gives a way to build some predefined database. For example to build the regional network of Chicago we use the instruction `%net=TrafficExample('Chicago')`. Then `%net` is visualized by the instruction `ShowNet()` which show the following graph in an editable graphic window of Scilab.

3. Traffic Assignment

3.1. **Wardrop Equilibrium.** Given a transportation network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ and a set $\mathcal{D}$ of transportation demands from $p \in \mathcal{N}$ to $q \in \mathcal{N}$ the traffic assignment problem consists in determining the flows f_a on the arcs $a \in \mathcal{A}$ of the network when the times t_a spent on the arcs a are given functions of the flows f_a. These flows are defined as solution of equilibrium. We can distinguish the deterministic equilibriums from the stochastic ones.

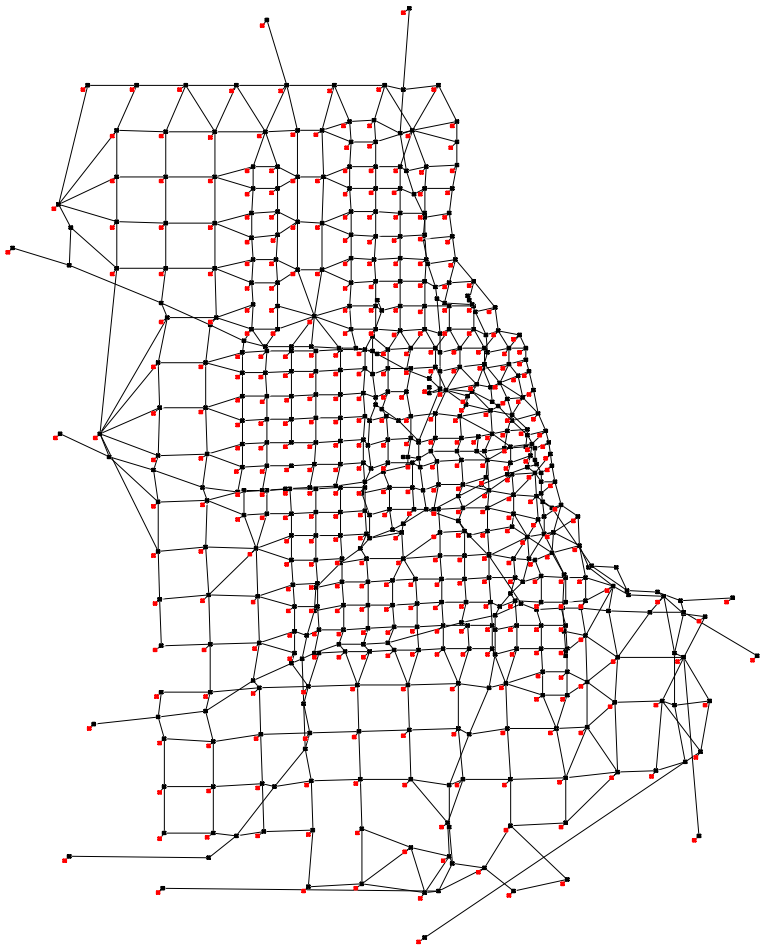

FIGURE 2. Chicago Net

In the deterministic case two important equilibriums have been defined. The system equilibrium where the flow are determined as minimizing the total time spent by all the users of the network. The user equilibrium, called sometimes Wardrop equilbrium where each user minimizes its time spent in the network; that has for consequence that at the equilibrium we have : for all pair of nodes p and q in the network, the time spent on all the actually used routes from p to q are the same and is smaller than the time spent on the unused routes.

Denoting by $\mathcal{C}$ the set of origin-destination pairs, R_{pq} the set of routes from p to q, d_{pq} the demand from p to q, f_r the route flow on the route $r \in R_{pq}$, F_a the arc flow on $a \in \mathcal{A}$, t_r the travelling time on the route $r \in R_{pq}$, t^*_{pq} the smallest travelling time of the routes in R_{pq}, the *Wardrop Equilibrium* is defined precisely by the following equations

$$f_r(t_r - t^*_{pq}) = 0\,, \quad t_r - t^*_{pq} \geq 0\,, \quad f \geq 0\,,$$

$$t^* \geq 0\,, \quad \sum_{r \in R_{pq}} f_r = d_{pq}\,, \quad \forall r \in R_{pq},\ \forall pq \in \mathcal{C}\,.$$

This system of equations is the optimality condition of two optimization prolems, the nodes-arcs and the arcs-routes variational formulation. The *nodes-arcs variational formulation* of the Wardrop Equilibrium is :

$$\min_f \sum_a \int_0^{F_a} t_a(q)\, dq, \quad A f_{pq}^{\bullet} = d_{pq}, \quad f \geq 0 \, .$$

where A is the $m \times n$ incidence matrix nodes-arcs associated to the network, f_{pq}^a is the flow on the arc a of the commodity pq, $F_a = \sum_{pq} f_{pq}^a$ is the total flow on the arc a and $t_a(q)$ denotes the time spent on the arc if the total flow on this arc is q.

The *arcs-routes variational formulation* of the Wardrop Equilibrium

$$\min_f \sum_a \int_0^{F_a} t_a(q) dq \, , \; F_a = \sum_{r \in R^a} f_r$$

$$\sum_{r \in R_{pq}} f_r = d_{pq} \, , \; f \geq 0 \, ,$$

where R^a denotes the set of all the routes of the network using the arc a.

The main drawback of this Wardrop equilibrium is that each traveler is supposed to have a perfect information on the total network. In more realistic formulations the user minimizes the perceived travel time. This perceived travel time is the effective travel time with an additive error. If the probabilistic distribution of the errors is supposed to be known we can define stochastic equilibriums.

3.2. **Logit Assignment.** At least two kind of stochastic equilibrium has been studied intensively in the literature according to the distribution of the error in the perceived travel time. In the *probit assignment* problem the error is supposed to be Gaussian. But, because $\mathcal{N}\{X > x\}$ for $\mathcal{N}$ a Gaussian law, is not known explicitely the computation of probit assignment is difficult and is done by Monte Carlo methods. When the distribution of the error is a Gumbel distribution $\mathcal{G}\{X < x\} = e^{-e^{-\mu x}}$ the probability to choose a route $r \in R_{pq}$ can be computed explicitly. It is given by

$$\forall r \in R_{pq} \;\; \mathbb{P}\{r\} = \frac{e^{-\mu t_r}}{\sum_{r \in R_{pq}} e^{-\mu t_r}} \, , \tag{1}$$

where t_r denotes the time spent on the route r. This assignment is called *logit assignment*. The only interest of the Gumbel distribution comes from the facility to compute the probability of the maximum of two independent random variables and from its shape close to the normal law distribution.

Another, more interesting justification of the logit assignment comes from that it is the only distribution satisfying the efficiency principe [12] that is : *"an event of flows on the available road having a smaller average time than another one has a greater probability to appear"*. In [12] other equivalent properties of the logit distribution (which is in fact the Gibbs distribution of mechanical satistics). In particular it minimizes the entropy among all the flow distributions having the same average time. The free parameter μ is a degree of stochasticity. Its inverse is the temperature in statistical mechanics.

The first difficulty appearing with computing logit assignment is that, as soon as it exists a circuit in the graph, it will exists a pair pq such that the number of routes in R_{pq} is infinite and therefore it may be difficult to compute the logit distribution(1). Dial [4] propose to consider only efficient paths r that are paths such that the time from p_i to q is decreasing with the nodes p_i met along the path $r = pp_1p_2\cdots q$ that is $t_{p_i\cdots q} < t_{p_{i+1}\cdots q}$ forall i in the path and from p to p_i are increasing. Then denoting $\mathcal{G}^{pq} = (\mathcal{N}, \mathcal{A}^{pq})$ the new graph obtained by eliminating the arcs for which this increasing and decreasing property is not satisfied and $\mathcal{O}^{pq}$ [resp. $\mathcal{D}^{pq}$ thecorrespondant incidence node-arc-origine [resp. node-arc-destination] matrix we can define the *transition weight matrix*

$$W^{pq} = \mathcal{O}^{pq} T (\mathcal{D}^{pq})' ,$$

where T is diagonal matrix $T_{aa} = e^{-\mu t_a}, \ \ a \in \mathcal{A}^{pq}$, We have

$$W^{pq}_{ij} = T_{aa}, \ \ \forall a \in \mathcal{A}^{pq} .$$

It exists a node numbering such that the matrices W^{pq} are strictly upper diagonal and therefore they are nilpotent. Thus $(W^{pq})^*$ exists where $N^* = \sum_{i=0}^{\infty} N^i$. Then the logit assignment on efficient path is given by :

$$F_{ij} = \sum_{pq \in \mathcal{D}} d_{pq} (W^{pq})^*_{pi} W^{pq}_{ij} (W^{pq})^*_{jq} / (W^{pq})^*_{pq} . \tag{2}$$

Indeed

- $(W^{pq})^*_{pq}$ is the total weight of all the efficient path from p to q,
- $(W^{pq})^*_{pi} W^{pq}_{ij} (W^{pq})^*_{jq}$ is the total weight of the efficient paths that use the arc ij,
- $(W^{pq})^*_{pi} W^{pq}_{ij} (W^{pq})^*_{jq} / (W^{pq})^*_{pq}$ is the probability for a path from p to q to use the arc ij for the logit distribution on the efficient paths.

In fact, Dial in [4] does not give the formula (2) but an algorithm in two phases to compute the contribution of one demand to the flow on an arc. Mainly in the first phase he computes $(W^{pq})^*_{pi} W^{pq}_{ij}$ in the second phase he computes $(W^{pq})^*_{jq} / (W^{pq})^*_{pq}$ from which it deduces the flow. Moreover the exact weights used by Dial are not the ones given here but the weights $e^{-\mu \tau_{ij}}$ with τ_{ij} defined by (5).

A first improvement, given in [8], consists in considering as efficient paths the paths which have an increasing time from the origin and not restricting them to have also a decreasing time to the destination. With this new definition of efficient paths we build new graphs $\mathcal{G}^p = (\mathcal{N}, \mathcal{A}^p)$, incidence matrices $\mathcal{O}^p$, $\mathcal{D}^p$ transition weight matrix W^p. The corresponding flow has a formula analogous to (2) but now can be written

$$F_{ij} = \sum_{p \in \mathcal{N}} (W^p)^*_{pi} W^p_{ij} (W^p)^*_{j\cdot} D^p \tag{3}$$

with

$$D^p_q = \begin{cases} d_{pq} / (W^p)^*_{pq} & \text{if } pq \in \mathcal{D} \\ 0 & \text{else} \end{cases} .$$

This formula shows that by this way the complexity can be reduce of one order in the size of the problem.

Another improvement proposed by Bell [1] is to consider all the paths, not only efficient paths, by remarking that we have not to enumerate all the paths but to be able to compute star of matrices. But $W^* = (I - W)^{-1}$ is well defined as soon as the spectral radius of W, denoted $\rho(W)$, is smaller than 1. Moreover with this point of view it is not necessary to compute the minimal time from the origine to each node to define the transition weight matrix associated to efficient paths. Therefore, we have, denoting by W the transition weight matrix for the original graph $\mathcal{G}$:

$$F_{ij} = \sum_{p \in \mathcal{N}} (I - W)^{-1}_{pi} W_{ij} (I - W)^{-1}_{j.} D^p \tag{4}$$

with

$$D^p_q = \begin{cases} d_{pq}/(I - W)^{-1}_{pq} & \text{if } pq \in \mathcal{D} \\ 0 & \text{else} \end{cases} .$$

With this point of view we must choose μ large enough such that $\rho(W) < 1$ (this is always possible because the entries of W are positive and go to zero when μ goes to $+\infty$).

We can avoid this last difficulty by building a Markov chain on $\mathcal{G}$ depending of a parameter μ whose trajectories converge to the minimal time trajectories when μ goes to $+\infty$. Let us consider the transition matrices

$$M^q_{ij} = e^{-\mu\tau_{ij}} / \sum_j e^{-\mu\tau_{ij}} ,$$

with

$$\tau_{ij} = t_{ij} + t^*_{iq} - t^*_{jq} , \tag{5}$$

where t^*_{iq} denotes the minimal time to go from i to q. This give a logit type probability to deviate from the optimal trajectory. Therefore here we consider a multidecision process where at each step we have to choose between trajectories composed of an arbitary admissible link followed by an optimal route to node q. Based on this type of decision the flows on arcs can be computed explicitely. We have

$$F_{ij} = \sum_q D_q (I - M^{\natural q})^{-1} M^{\flat q} M^q_{ij} , \tag{6}$$

where D_q is the row-vector of the demand from any node to q, $M^{\natural q}$ denotes the matrix obtained by skipping column an row q in matrix M^q and $M^{\flat q}$ the column-vector obtained by extracting column q of M^q and skiping entry q.

3.3. **Stochastic Equilibrium :** `MSA, LogitNE, MSASUE.` In the previous section about stochastic assignment we have supposed implicitely that the travelling time does not depend on the flow. If it is not the case we have to solve an implicit equation. For example in the third case, given the function $t_a(F_a)$ for $a \in \mathcal{A}$ we have to solve the equation

$$F = \mathcal{F}(F) \tag{7}$$

with

$$[\mathcal{F}(F)]_{ij} = \sum_{p\in\mathcal{N}} (I - W(F))^{-1}_{pi} W(F)_{ij} (I - W(F))^{-1}_{j\cdot} D^p(F) \ ,$$

where $W(F)$ is the weight transition matrix which depend on the flows.

This equilibrium admits an arc-route variational formulation :

$$\min_f \sum_r f_r \log(f_r) + \sum_a \int_0^{F_a} t_a(q) \ , \tag{8}$$

under the constraints :

$$F_a = \sum_{r\in R^a} f_r, \ \sum_{r\in R_{pq}} f_r = d_{pq}, \ f \geq 0 \ .$$

The fixed point algorithm

$$F^{n+1} = a_n F^n + (1 - a_n)\mathcal{F}(F^n) \ ,$$

with typically $a_n = (n-1)/n$, is used to solve (7). The convergence of this method can be derived from the variational formulation. Indeed $\mathcal{F}(F^n)$ is the solution of

$$\min_f \sum_r f_r \log(f_r) + \sum_a t_a(F^n_a)(F_a - F^n_a) \ ,$$

which is a partially linearized version of problem (8). Then using the convexity of this last problem $\mathcal{F}(F^n) - F^n$ is a descent direction of problem (8). The divergent series method is used because it is difficult to compute the value of the criterium indeed it needs the evaluation of the flows on all the routes (which may be in infinite number).

3.4. **Assignment in CiudadSim.** The assignment is done by the instruction `TrafficAssig()` see, in the manual the `TrafficAssig` section below, the exhaustive list of algorithms and methods of assignment available. The default algorithm is the `'DSD'` which is a newton method to compute a Wardrop equilibrium. We can change the algorithm by changing `%net.gp.algorithm`. If we choose a logit algorithm we have to specify the level of stochasticity by changing the default value of `%net.gp.theta`. A discussion and comparison of the respective interests of the algorithm is done in the section "Numerical Experiments".

4. Assignment Examples

To see the exhaustive list of examples available look at the section `Traffic-Example` of the manual. In the following is given only a sampling of the possibility. Typically to make an assignment for an existing example we have to write the following instructions in the Scilab window :

```
%net=TrafficExample('Example')
TrafficAssig()
ShowNet()
```

where `'Example'` is the name of an available example. If a new example has to be entered the best is to build `%net` with `MakeNet`. It is also possible to edit a new net with the Scigraph window.

4.1. **Regular City.** The instruction

```
%net=TrafficExample('Regular City',hs,vs,nd)
```

generates a regular city whose graph is a `hs` x `vs` grid. The link travel time functions data is the same for all links and is defined by the parameters `t0=ca=ma=ba=1` and the link performance model 3 therefore link performance function is $c = t_0(1 + 0.15f)$.

It is possible to generate 4 different types of Regular City depending on parameter `nd`.

(1) `nd=[ ]` The net has all possible O-D pairs all of them with demand value equal to 1. The first picture in figure (3) is shown using the

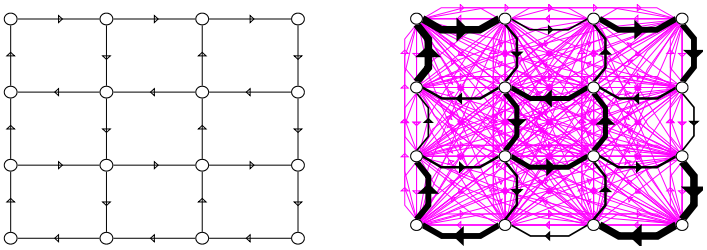

FIGURE 3. Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',4,4)`

function `ShowLinks` because there are to many demands.

(2) `nd=1` There is only one O-D pair joining the most distant nodes with demand equal to 1.

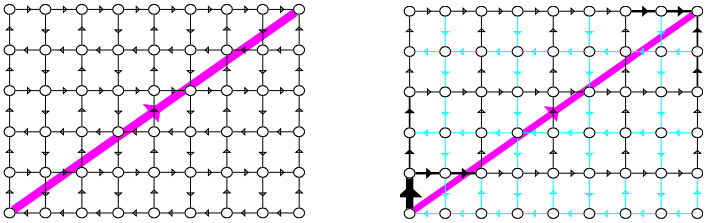

FIGURE 4. Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',6,9,1)`

(3) `0<nd<1` The parameter nd represents the density of O-D pairs present in the net and the demand is equal to 1.

(4) `1<nd` There are `nd` random demands generated with value uniformly distributed between 0 and 15.

4.2. **Sioux Falls Net.** It is an example with 24 nodes 72 links and 528 demands with link performance function of the form $c = t_0 + m_a f^4$ with t_0 between 0.02 and 0.1 and m_a between 10^{-8} and 2×10^{-5}.

4.3. **Steenbrink Net.** The SteenBrink example can be found in: Steenbrink P.A., *Optimization of Transport Networks*, John Wiley & Sons, London, 1974. It is an example with 9 nodes 36 links and 12 demands with link performance function of the form $c = t_0 + m_a f$ with t_0 between 0 and 1 and m_a between 3×10^{-4} and 2×10^{-3}.

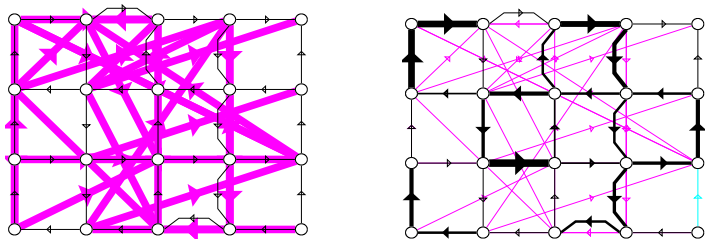

FIGURE 5. Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',4,5,0.09)`

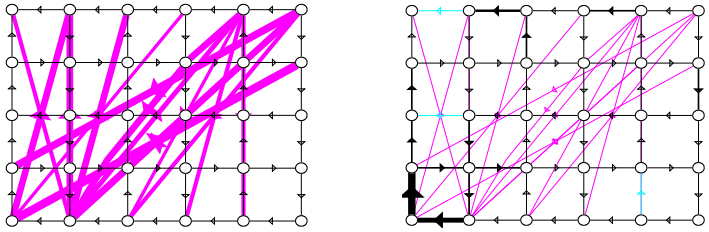

FIGURE 6. Non-assigned and Assigned Regular City Net `TrafficExample('Regular City',5,6,15)`

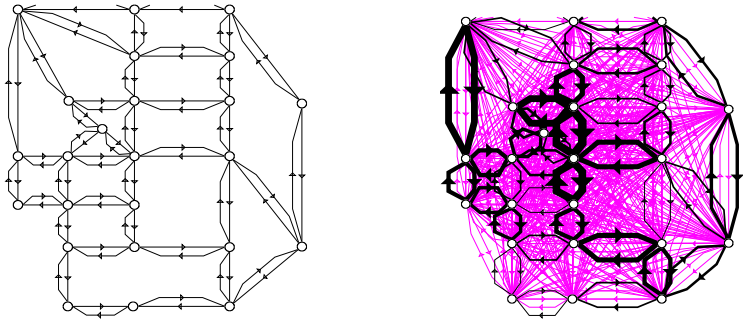

FIGURE 7. Non-assigned and Assigned Sioux Falls Net `TrafficExample('Sioux Falls')`

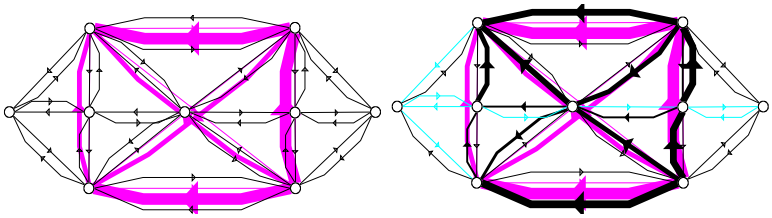

FIGURE 8. Non-assigned and Assigned Steenbrink Net `TrafficExample('Steenbrink')`

5. ALGORITHMS

Let us describe briefly the algorithm used to compute Wardrop algorithm which have been implemented. Logit assignment correspond to explicit formula given in the section "Traffic Assignment" the corresponding

equilibrium uses a method of successive average described in the subsection "Stochastic Equilibrium" the corresponding algorithm are implemented in the functions

- `MSA` method of successive average in deterministic case,
- `LogitNE` method of successive average in logit case,
- `MSASUE` method of successive average in probit case.

5.1. **All or Nothing algorithm : `AON`.** When the times spends on the arcs do not depend of the flows on the arcs, the problem may be reduced to compute the routes with smallest the travel time from each pair pq in $\mathcal{C}$.

5.2. **Incremental Loading Heuristic : `IA`.** The demand is splitted in a sum of elementary demands. Each elementary demand is assigned using the AON algorithm after a fresh evaluation of the travelling time based on the previous loadings.

The assignment obtained by this method does not correspond in general to a Wardrop Equilibrium.

5.3. **Newton Method for the arcs-nodes variational formulation : `WardropN, ('NwtArc').`** Using the nodes-arcs formulation, introducing the dual variable V associated to the conservation law written $AF = D$, and denoting $1/2F'RF$ the quadratic criterium to be minimized, we obtain the optimality conditions :

$$AR^{-1}\max(A'V,0) = D\,,\quad F = R^{-1}\max(A'V,0)\,.$$

As soon as the network is strongly connected the first equation defines V up to a constant. It can be computed efficiently by a Newton method which is not globally convergent. F is unique and may be deduced from V using the second equation.

Using relaxation algorithm this method had been adapted for general costs to the multicommodity case.

It seems to be efficient when the number of commodities is not very large and when the flow depending part of the travel time dominates its fixed part.

5.4. **Frank-Wolfe algorithm : `FW`.** The nodes-arcs formulation is used.

The current assignment is improved according to :

- Given the flows F_a, the travel times $t_a(F_a)$ are computed and a AON assignment F' is done.
- The new assignment becomes $\lambda F + (1-\lambda)F'$:

$$\min_{1\geq\lambda\geq 0}\left(\sum_a c_a\left(\lambda F_a + (1-\lambda)F'_a\right)\right),\quad c_a(F_a) = \int_0^{F_a} t_a(q)\,dq\,.$$

It is the most common used algorithm. It may be slow because it is a first order method but it is not very memory consuming therefore it can be applied to large networks.

5.5. **Disaggregated Simplicial Decomposition : `DSD`.** This algorithm have been proposed by Larsson-Patriksson [6].

The nodes-routes formulation is used.

- At starting time, all the demand of a commodity is assigned to only one route by an AON algorithm.
- At each iteration, for all commodity, the shortest route (based on the current flow) is added to a set of memorized routes.
- The improved assignment is obtained by optimizing the flow on the memorized routes by a diagonalized Newton method.

Using the variables λ_{pqr} which gives the proportion of the demand assigned to the route r for the commodity pq we have

$$f_r = \lambda_{pqr}\, d_{pq}\,.$$

the arc flow on a is :

$$F_a = \sum_{pq} d_{pq} \sum_{r \in R^a_{pq}} \lambda_{pqr}\ .$$

The cost function becomes :

$$T(\lambda) = \sum_{a \in A} c_a \left(\sum_{pq} d_{pq} \sum_{r \in R^a_{pq}} \lambda_{pqr} \right)\ .$$

To compute the descent direction μ a second order approximation of the cost T is made:

$$T(\mu) \approx T(\lambda) + DT(\lambda)(\mu - \lambda) + (\mu - \lambda)'\ D^2T(\lambda)\ (\mu - \lambda)\ ,$$

where

$$[DT(\lambda)]_{pqr} = d_{pq} \sum_{a \in r} c'_a(F_a)\,, \quad [D^2T(\lambda)]_{pqr,p'q'r'} = d_{pq}\, d_{p'q'} \sum_{a \in r \cap r'} c''_a(F_a)\ .$$

Then the new assignment is obtained by optimizing the quadratic diagonal approximation of the cost, that is taking only the diagonal part of the Hessian :

$$[D^2T(\lambda)]_{pqr,pqr} = d^2_{pq} \sum_{a \in r} c''_a(F_a)\ .$$

and solving the linear search :

$$\min_{\rho \geq 0} T(\lambda + \rho\mu)\ ,$$

where λ denotes the current assigment.

6. Numerical Experiments

All the implementations of the algorithms for this study where on a Pentium III 448.623 MHz computer with 384 MB of internal memory and a 7.4 GB hard disk. The algorithm were tested on networks known from the literature and over regular network of different sizes. In the tables time is expressed in seconds.

6.1. **Deterministic Case.** All the numerical examples shown here have some common properties. The best values for the objective function are reached with the DSD algorithm and at the same time it is the most memory demanding algorithm because it is a route-based algorithm and in each step it is store route information using the back node arrays generated in the shortest path procedure. The CPU time required depends on the considered example.

The first example is the Steenbrink network. We observe here that DSD is the most performat algorithm, it arrives to a better value of the cost function in a time which is largely smaller than the time needed by the other three algorithms.

TABLE 1. Steenbrink, nodes=9, links=36, demands=12

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	5	0.29	1.6957675e+04	2.7e+03	8.245e-05
FW	4325	38.75	1.6958960e+04	2.6e+04	9.995e-05
MSA	4494	19.89	1.6959068e+04	2.2e+04	9.853e-05
NwtArc	30	13.80	1.6957675e+04	2.1e+03	6.475e-05

The second example is the well known numerical example of Sioux Falls. In this case it is not possible to apply the NwtArc algorithm. Here again DSD is the best algorithm taking into account computation time and accuracy, but it is also the most memory demanding algorithm because it is a route-based algorithm and in each step it is store route information using the back node arrays generated in the shortest path procedure.

TABLE 2. Sioux Falls, nodes=24, links=72, demands=528

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	10	1.58	4.2313361e+01	6.3e+04	7.656e-05
FW	1789	23.48	4.2316005e+01	1.4e+04	9.999e-05
MSA	13933	181.24	4.2317409e+01	7.1e+04	9.999e-05

The following problem is a slightly different version of Chicago Sketch.

TABLE 3. Chicago, nodes=546, links=2176, demands=93135

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	3	127.40	1.6753004e+07	4.7e+06	5.499e-04
FW	33	153.40	1.6753136e+07	4.9e+05	5.781e-04
MSA	137	408.09	1.6757105e+07	1.9e+05	5.489e-04

We can see in the following example the performance of the 4 available algorithms to make the assignment in the case of few OD pairs. In the first table the difference between the values of the cost function for each algorithm are smaller than $1.7e-5$ but the difference in the computational times is significant.

The algorithm 'NwtArc', is the most performance one in the case of a unique OD pair but it becomes less efficient increasing the demand number

TABLE 4. Regular City, nodes: 225, links:420, demands:1

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	28	3.14	2.854915580e+01	1.4e+04	1.353e-04
FW	220	12.55	2.854917233e+01	3.7e+03	5.011e-05
MSA	258	2.86	2.854916956e+01	3.4e+03	9.140e-05
NwtArc	3	0.63	2.854917270e+01	9.3e+03	2.395e-08

as we can see in the following table. The computation are made in a 10×10 regular network.

TABLE 5. Performance of the NwtArc algorithm

Demand Number	Algo	Step	Time	Cost	Memory Used	Relative Gap
1	DSD	11	0.59	1.8470456e+01	5.1e+03	6.567e-05
	NwtArc	2	0.24	1.8470467e+01	4.6e+03	1.917e-06
2	DSD	14	0.92	3.7870531e+01	5.6e+03	5.010e-05
	NwtArc	6	0.71	3.7870529e+01	4.8e+03	8.141e-05
10	DSD	12	1.27	1.6139485e+02	1.4e+04	4.777e-05
	NwtArc	8	4.49	1.6139488e+02	5.7e+03	8.415e-05
100	DSD	13	5.56	2.1215788e+03	7.2e+04	5.197e-05
	NwtArc	15	97.33	2.1215787e+03	1.6e+04	3.331e-04
900	DSD	15	27.12	6.403253421e+04	2.9e+05	5.112e-04
	NwtArc	15	1105.02	6.403153657e+04	1.0e+05	2.870e-04
1000	DSD	15	31.61	7.607687457e+04	3.2e+05	8.147e-04
	NwtArc	15				

To illustrate the performance of the DSD algorithm compared to FW and MSA algorithm, we present results of the algorithms applied to a regular 8×8 network, whose sizes are: 64 nodes, 112 links, and 4032 OD pairs. The following problem is a 8×8 regular network. .

TABLE 6. Regular, nodes=64, links=112, demands=4032

Algorithm	Iter	Time	Cost	Mem. used	Rel. Gap
DSD	6	135.67	5.0910377e+05	3.02e+05	1.2e-05
FW	3000	158.14	5.0911997e+05	2.7e+04	5.3e-05
MSA	3000	136.28	5.0912299e+05	2.4e+04	5.1e-05

In figure (9), the logarithm of the relative error obtained for each algorithm are plotted again the CPU time also in logarithmic scale. The relative errors are computed with reference to the best value found for the cost function with the DSD algorithm with a precision of 10^{-9}. The appearance of figure (9) is the expected one, since FW and MSA are both first order algorithms so they have a similar behavior decreasing like $1/t$ while the second order algorithm, DSD, decrease as a power of $1/t$.

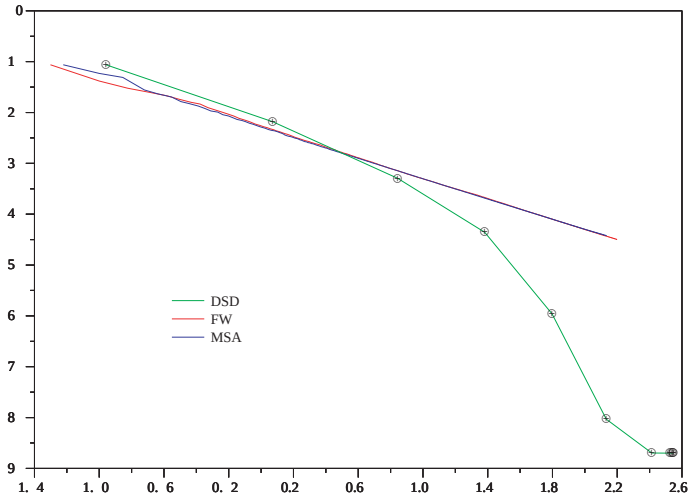

FIGURE 9. log(Rel. error) vs log(time) for the FW, MSA and DSD algorithms

6.2. **Stochastic Case.** For the case of stochastic assignment, we present here some results concerning the relation between the different type of logit algorithms. We consider the example of a 6×6 regular network with all possible OD pairs. In table (7) we show the results for the computation of the stochastic equilibrium. If parameter μ is very large, the perception error

TABLE 7. Stochastic Equilibrium: logit type algorithm comparison

Algorithm	Iter	Time	Cost	Mem used	Error
Logit DE	147	41.27	6.2391744e+04	1.2e+05	9.922e-04
Logit BE	147	1.67	6.2391775e+04	1.2e+05	9.923e-04
Logit MDE	132	36.41	6.2390267e+04	1.2e+05	9.921e-04
Logit MBE	132	34.52	6.2390281e+04	1.2e+05	9.921e-04

is small and users will tend to select the minimum measure travel-time path, if μ is small the share of flow on all paths will be equal. We can see this in figure (10) where the computation were made with a logit-dial algorithm . The left graphic represents the flow assignment obtained with a $\mu = 1e+3$, this flow distribution is the same that is found using the AON algorithm. The graphic in the right is obtained with $\mu = 1e-3$, all efficient path are used and the flow is uniformly distributed among those paths.

In next example we consider a regular city with a unique demand, in this case all the possible paths joining the origin and destination nodes have the same cost. If we performe an assignment using the AON algorithm, all the demand flow is assigned to one path. In the other hand if a logit-type algorithm is used all the path of equal cost are used.

Finally we present the results of computing the stochastic equilibrium for a large value of parameter μ with the method of successive average and a bell-type logit, the obtained flow distribution is very similar to the distribution obtained by a deterministic assignment performed with the DSD algorithm.

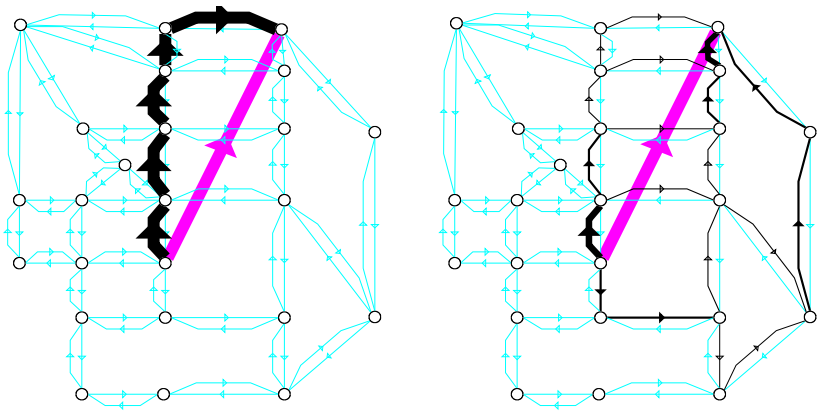

FIGURE 10. Logit assignment with $\mu = 1e+3$ and $\mu = 1e-3$

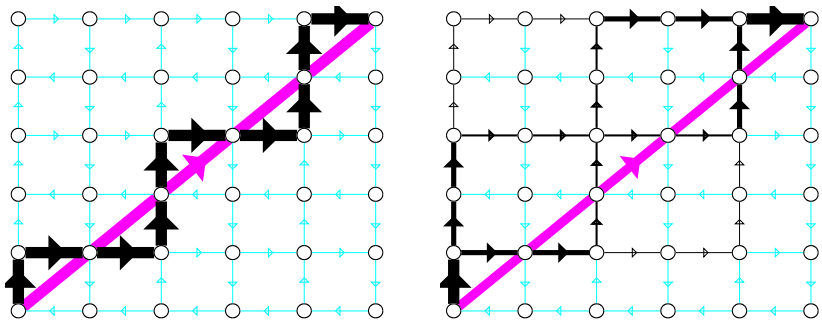

FIGURE 11. AON and Logit assignment in a regular network

The maximum difference flow value is of the same order than the required precision (See figure 12).

Algorithm	Iter	Time	Cost	Mem used	Error
Logit BE	147	1.67	6.2391775e+04	1.2e+05	9.923e-04
DSD	3	2.08	6.2379125e+04	5.6e+04	3.666e-05

TABLE 8. Stochastic and deterministic equilibriums

7. CiudadSim Manual

We give here by alphabetic order all the function and data structures of CiudadSim. All this documentation is available inline in ascii (by the help of scilab) and html form in the directory man of CiudadSim.

It is useful also to make hierarchy in these functions. The high level ones that a user has to know to make an assignment and the low level used by the high level and useful for developpers

The high level functions and data structures are

- `Netlist` contains a complete description of the structure of the database.
- `TrafficExample` to build some example of database.

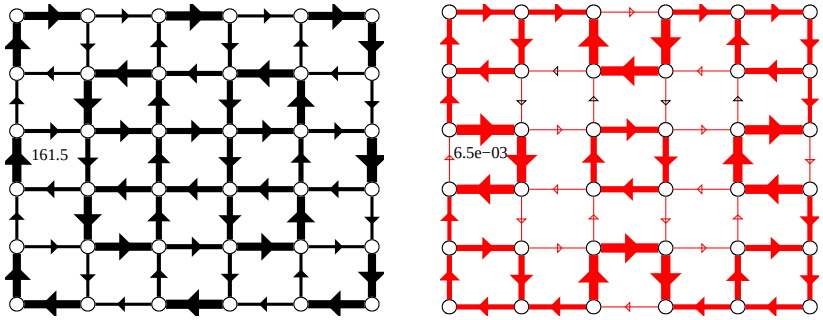

FIGURE 12. Stochastic equilibrium assignment and difference with a deterministic equilibrium assignment

- `AddLinks, AddNodes, AddDemands` to edit by Scilab programming the database.
- `TrafficAssig` to compute the assignment with various algorithms for various modelling.
- `ShowNet, ShowLinks, ShowDemands` to visualize the datas or the assignment.
- `ExportMI, MI2Scilab` to export a net to Mapinfo and to import a net from Mapinfo.

All the other functions are low level ones defining assignment algorithms, facilities or useful intermediary computations.

7.1. **AON** — ALL OR NOTHING TRAFFIC ASSIGNMENT ALGORITHM.

CALLING SEQUENCE :

```
[F]=AON(net)
[F]=AON(net,t0)
[F]=AON(tl,hl,t0,nn,origins,td,hd,dd)
```

PARAMETERS :

`net` : a NetList,
`t0` : row vector, the link travel time (if not given in net)
`tl, hl` : row vectors, tail and head nodes numbers of the links
`nn` : node number
`origins` : origins of the OD pairs
`td, hd, dd` : row vectors, tail and head nodes numbers of the demands, and demands values
`F` : assigned flow

DESCRIPTION :
Assigns the flow with the All or Nothing algorithm. For each OD-pair it looks for the minimum travel time path and assigns all the demand to this path. The travel time of the links are given by `t0` or by the corresponding field in net `net (net.links.t0)`. The non feasible OD pairs are ignored.

EXAMPLES :

```
net=TrafficExample('Small');
F=AON(net)
```

SEE ALSO : AONd 19, IA 24, IAON 25, FW 23, CAPRES 21, TrafficAssig 43

7.2. AONd — ALL OR NOTHING TRAFFIC ASSIGNMENT ALGORITHM.

CALLING SEQUENCE :

```
[F,f]=AONd(net)
[F,f]=AONd(net,t0)
[F,f]=AONd(tl,hl,t0,nn,origins,td,hd,dd)
```

PARAMETERS :

net : a NetList,
t0 : row vector, the link travel time (if not given in net)
tl, hl : row vectors, tail and head nodes numbers of the links
nn : node number
origins : origins of the OD pairs
td, hd, dd : row vectors, tail and head nodes numbers of the demands, and demands values
F : assigned flow
f : assigned flow disaggregated by commodity

DESCRIPTION :

Displays the non feasible OD pairs, if there are some. Those pairs are ignored in the assignment. Assigns the flow with the All or Nothing algorithm. For each OD-pair it looks for the minimum travel time path and assigns all the demand to this path. The travel time of the links are given by t0 or by the corresponding field in net (net.links.t0).

EXAMPLES :

```
%net=TrafficExample('Nguyen Dupuis');
// we can see the Net
ShowNet();
//we add a non feasible OD pair
%net=AddDemands(%net,8,1,10);
ShowNet()
[F,f]=AONd(%net);
```

SEE ALSO : AON 18, IA 24, IAON 25, FW 23, CAPRES 21, TrafficAssig 43

7.3. AddDemands — ADD DEMANDS TO A NET LIST.

CALLING SEQUENCE :

```
AddDemands(td,hd,demand)
```

PARAMETERS :

%net : global Netlist variable database

`td` : row vector, tail nodes numbers of the added demands
`hd` : row vector, head nodes numbers of the added demands
`demand` : row vector, the values of the added demands

DESCRIPTION :
`AddDemands` adds demands to `%net` (the database).

EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add a new demand from node 3 to node 2 with value 10
//    in the  net
AddDemands(3,2,10);
// We can see the modifications with
ShowNet()
```

See Also : `IntroTrfAsg` 26, `NetList` 35, `AddLinks` 20, `AddNodes` 20, `MakeNet` 34, `ShowNet` 42

7.4. AddLinks — Add links to a net list.

CALLING SEQUENCE :

```
AddLinks(tl,hl,lpp)
```

PARAMETERS :

`%net` : The global Netlist variable database
`tl` : row vector, tail nodes numbers of the added links
`hl` : row vector, head nodes numbers of the added links
`lpp` : 4 x nl matrix, travel time function parameters, where nl is the number of added links

DESCRIPTION :
`AddLinks` adds links to `%net` (the database).

EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add a new link from node 3 to node 4 with lpp=[0;0;1;1]
AddLinks(3,4,[0;0;1;1]);
// We can see the modifications with
ShowNet()
```

See Also : `IntroTrfAsg` 26, `NetList` 35, `AddNodes` 20, `AddDemands` 19, `MakeNet` 34, `ShowNet` 42

7.5. AddNodes — Add nodes to a net list.

CALLING SEQUENCE :

```
AddDemands(nx,ny)
```

PARAMETERS :

`net` : a Netlist
`nx` : row vector, x coordinates of the added nodes
`ny` : row vector, y coordinates of the added nodes

DESCRIPTION :
`AddNodes` adds nodes to `%net` (the database).

EXAMPLES :

```
%net=TrafficExample('Diamond');
ShowNet()
// Add 2 new nodes with coordinates x=[346,346] and y=[559,-50]
AddNodes([346,346],[559,-50]);
// We can see the modifications with
ShowNet()
```

See Also : `IntroTrfAsg` 26, `NetList` 35, `AddDemands` 19, `AddLinks` 20, `MakeNet` 34, `ShowNet` 42

7.6. **CAPRES** Capres traffic assignment algorithm.

CALLING SEQUENCE :

```
[f,ta]=CAPRES(net)
```

PARAMETERS :

`f` : assigned flow (disaggregated by commodity)
`ta` : times for the current assigned flow
`net` : a NetList

DESCRIPTION :
Assigns the flow with the CAPRES algorithm. As in the IAON successive AON assignments are made, but the number of iterations is fixed to 4 and instead of using the current travel times, the old travel times are combined in the same way the flows are.

EXAMPLES :

```
net=TrafficExample('Small');
[f,t]=CAPRES(net);
```

See Also : `AON` 18, `IAON` 25, `IA` 24, `FW` 23, `TrafficAssig` 43

7.7. **DSD** Disaggregated Simplicial Decomposition algorithm.

CALLING SEQUENCE :

```
[F,ta,ben,f]=DSD(net,routemax,nFW,eps,list_eps,list_itemax)
[F,ta,f]=dsd(net,routemax,nFW,eps,list_eps,list_itemax)
```

PARAMETERS :

`net` : a NetList
`routemax` : maximum number of general iterations
`nFW` : number of Frank-Wolfe iterations
`eps` : general precision

`list_eps` : array of precisions, Newton method precision, auxiliar precision and linear search precision
`list_itemax` : array of maximum number of iterations, Newton iterations, auxiliar iteration and linear search iteration
`F` : assigned flow
`ta` : link travel time for the assigned flow f
`ben` : `nix5` matrix (`ni` number of performed global iterations) benchmark information
`f` : assigned flow disaggregated by commodity

DESCRIPTION :
Assigns the flow with the disaggregated simplicial decomposition algorithm. This algorithm generates at each iteration a new route for each commodity using AON. The traffic is assigned minimizing the total cost over the generated routes using a Newton method. The version `DSD` shows performance statistics. The version `dsd` is used mostly inside other algorithms and it does not displays any information.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Steenbrink');
// uses the example net Steenbrink
[F,ta,ben,f]=DSD(net,20,0,1e-4,[1e-6,1e-7,1e-8],[2,50,100]);
```

See Also : `AON` 18, `IAON` 25, `CAPRES` 21, `FW` 23, `TrafficAssig` 43

7.8. **ExportMI** — Scilab to MapInfo interface.

CALLING SEQUENCE :

```
ok=ExportMI(net,name,nameout,disp_option)
```

PARAMETERS :

`net:` Scilab variable containing the network
`name:` the name of the MI source file without extension
`nameout:` the name of the MapInfo table where the results will be saved
`disp_option:` displaying options.

DESCRIPTION :
This command is meant to display in MapInfo the results obtained for a network already recovered from mapinfo. It uses an existant MI file named name and creates two files : one .mif and one.mid.

To see them in MI , they must be imported (table/import), then the values obtained in Scilab will be in the table associated with the file, and the width of the arcs will be proportional to the chosen magnitude, flow or time.

The displaying option is used to specify the magnitude shown in the arcs. The values for `disp_option` are the strings `'flow'` and `'time'`.

EXAMPLE :

```
// 1) Choose the example Versailles.
// This network comes from  from a MapInfo file
%net=TrafficExample("Versailles");
// 2) Make an assignment
TrafficAssig()
// 3) Translate to mapinfo
ok=Scilab2MI(%net,CS_DIR+"/examples/SGL_Versailles_v",CS_DIR+..
                                        "/examples/man_example")
// 4) Look the content of the variable CS_DIR
//                             (you need it in MapInfo)
// 5) Open MapInfo and import the tables man_example,
// look for it in CS_DIR then in examples
```

See Also : ImportMI 25

7.9. FW Frank-Wolfe traffic assignment algorithm.

CALLING SEQUENCE :

```
[F,ta,ben]=FW(net,kmax,tol)
```

PARAMETERS :

net : a NetList
kmax : maximum number of iterations
tol : precision
F : assigned flow
ta : link travel time for the assigned flow F
ben : nix5 matrix (ni number of performed iterations) benchmark information

DESCRIPTION :
Assigns the flow with the Frank-Wolfe algorithm. After an initial AON assignment the link costs are updated and a new assignment is computed. The new assignment is combined with the previous one in a way that minimizes the global cost. The algorithm terminates when the number of iterations kmax is reached or the change in link costs is less than tol. When they are not given tol and kmax have the default values 1e-6 and 20.

The matrix ben contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
/// generates a network with 4 nodes, 9 arcs and 2 OD-pairs
[F,ta,ben]=FW(net);
```

See Also : AON 18, IAON 25, CAPRES 21, DSD 21, TrafficAssig 43

7.10. **Graph2Net** — Recovers the net from a graph.

CALLING SEQUENCE :

```
net=Graph2Net(g,nf)
```

PARAMETERS :

`g` : graph-list
`nf` : integer between 0 and 6, chooses the travel time formula
`net` : a NetList

DESCRIPTION :

Graph2Net recovers the NetList from a graph-list. In order to distinguish the demands and links arcs, the following color conventions must be satisfied in the graph:

- the links arcs must have color 1 (black),
- the demands arcs must have the color 6 (cyan).

After saving an edited traffic network (shown with `ShowNet`) in a scigraph window we get a graph-list that we can transform in a NetList with `Graph2Net`.

EXAMPLES :

```
%net=TrafficExample('Empty');
ShowNet()
// Now you can edit in the scigraph window following the
// color conventions  to  distinguish the demands  from
// the links and then save the graph.
g=load_graph('name.graph');
%net=Graph2Net(g); // By default nf=6
// You can check if the net is the same you saved.
ShowNet()
```

See Also : `IntroTrfAsg` 26, `TrafficExample` 45, `ShowNet` 42

7.11. **IA** — Incremental traffic assignment algorithm.

CALLING SEQUENCE :

```
[f,ta]=IA(net,k)
```

PARAMETERS :

`net` : a NetList
`k` : number of iterations of the algorithm
`f` : assigned flow (disaggregated by commodity)
`ta` : row vector of reals, current time for the assigned flow f

DESCRIPTION :

Assigns the flow with the Incremental Assignment algorithm. Each origin-destination flow is divided into `k` equal parts. Each part is assigned by `AON`. The link costs are updated using the travel time functions of the assigned flow `f`.

EXAMPLES :

```
net=TrafficExample('Small');
[f,ta]=IA(net,10);
```

See Also : AON 18, IAON 25, CAPRES 21, FW 23, TrafficAssig 43

7.12. IAON — Iterated All or Nothing traffic assignment algorithm.

CALLING SEQUENCE :

```
[f,ta,ben]=IAON(net,k)
```

PARAMETERS :

net : a NetList
k : number of iterations of the algorithm
f : assigned flow (disaggregated by commodity)
ta : row vector of reals, current time for the assigned flow
ben : a kx5 matrix benchmark information

DESCRIPTION :
Assigns the flow with the Iterated All or Nothing heuristic. Starting with a feasible flow assigned with AON the link costs are calculated with the travel time functions. The AON assignment is made now with the computed costs and then the costs are recomputed for the new assignment. This process is repeated until there is no change in the re-calculated costs or the number of iterations k is reached.

The matrix ben contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
[f,ta,ben]=IAON(net,10)
```

See Also : AON 18, IA 24, FW 23, CAPRES 21, TrafficAssig 43

7.13. ImportMI — MapInfo to scilab interface.

CALLING SEQUENCE :

```
[net,color,Bidir]=ImportMI(name)
```

PARAMETERS :

net : Netlist structure : the resulting network
color : a vector with the color of the links
Bidir : a sparse matrix

DESCRIPTION :
The purpose of the MapInfo interface is to facilitate the input of the geographical data into the CiudadSim Toolbox and to provide a nicer way of

displaying the results obtained after the assignment. There are two commands : ImportMI (to translate from MI to Scilalb) and ExportMI (to translate from Scilab to MI).

The MapInfo files must created, for that do:

- In MapInfo, open the file MI_empty.tab located in the directory examples of the MCiudadSim toolbox.
- Save it with another name to be able to modify it.
- Close the MapInfo table MI_empty and open the one you created.
- Make the layer editable.
- Press the F key to enter in the fusion mode
- Draw the arcs as polylines.
- Export the table to obtain two files one .mif and other .mid .

Important Remarks :

(1) The extreme of the arcs will be considered as nodes so if you want that two arcs share the same node is important to draw it like that. That's why the Fussion mode is important, when the pointer is over an already defined node it will change into a cross.

(2) The style of the polyline is used to differentiate one-way and two-ways links. The two-ways links can have any style but one-way links must have style 2. Each two-way link in MapInfo corresponds to two one-way links in Scilab. This correspondance is found in the sparse matrix Bidir.

(3) The links can be given different colors, these colors will be in the output as the vector color, it can be useful to differentiate subsets of links.

EXAMPLES :

```
// MapInfo translation from the files
//    - SGL_Versailles_v.mid
//    - SGL_Versailles_v.mif
// Regular town example with four modes and two classes.
[%net,color,Bidir]=ImportMI(CS_DIR+..
                          "/examples/SGL_Versailles_v");
// Show the translated arcs in Scilab
ShowNet()
```

See Also : ExportMI 22

7.14. **IntroTrfAsg** —— Introduction to the Traffic Assignment Toolbox.

DESCRIPTION :

The purpose of the toolbox is to solve traffic assignment problems. A good reference is Michael Patriksson's book "The Traffic Assignment Problem, Models and Methods".

To define the problem we need :

(1) The Traffic Network : an oriented graph G where `nn` nodes represent places or cities, and `na` arcs represent links or roads.

(2) The Link Performance Functions (also known as travel time functions): the travel time as a function of the flow in the arc. Seven formulae are used needing 4 parameters. Once a formula is chosen for the whole network, the parameters are defined in a `4×na` matrix.
(3) The Origin-Destination traffic demand pairs (OD-pairs also called commodities) are given by an initial node, a final node and a flow. The `nd` OD-pairs are defined by a `3×nd` matrix.

All the data are stored in a scilab global variable `%net` which is a structure called a `NetList` that plays the role of a geographic data basis which can be visualized by `ShowNet` and edited inside "metanet" or "scigraph". This NetList is build by `MakeNet`.

The traffic assignment is computed by `TrafficAssig`. The results are stored in the Geographic Data Basis and can be visualized with `ShowNet`.

See Also : `NetList` 35, `MakeNet` 34, `ShowNet` 42, `TrafficAssig` 43

7.15. **LogitB** — Logit equilibrium (Bell Method).

CALLING SEQUENCE :

```
[F]=LogitB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :

Compute the logit traffic assignment by the Bell method where all the routes are considered and not only the efficient ones (see `LogitD`).

To each route r on the graph defined by the matrix `A` is associated a weight $e^{-\theta l}$ with $\theta =$ `theta` where l denotes the the total travel time on the route r. The flow for each demand defined by the matrix `D` is assigned on a route proportionally to its weight.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

The variable `theta` must be large enough to insure that the weight of routes with an infinite number of links are finite.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected),
// n is the number of nodes, m the number of arcs.
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
```

```
theta=10;
// Flow calculation
FD=LogitB(A,D,theta)
```

See Also : LogitD 28, LogitMB 29, LogitMD 29, LogitN 30, LogitNE 31, TrafficAssig 43

7.16. LogitD — Logit equilibrium (Dial Method).

CALLING SEQUENCE :

```
[F]=LogitD(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Dial method where efficient routes are the ones which never go back towards the origin.

To each efficient route r on the graph defined by the matrix A is associated a weight $e^{-\theta l}$ where l denotes the the total travel time on the route r. The flow for each demand defined by the matrix `D` is assigned on a route proportionally to its weight.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected),
// n is the number of nodes, m the number of arcs.
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitD(A,D,theta)
```

See Also : LogitB 27, LogitMB 29, LogitMD 29, LogitN 30, LogitNE 31, TrafficAssig 43

7.17. **LogitMB** — Logit equilibrium (Markov Bell Method).

CALLING SEQUENCE :

```
[F]=LogitMB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows
`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Markov method where on each road at each crossing we have to choose a link among the links leaving this crossing including the ones coming back to the origin.

A Markov chain is defined by normalizing the following transition matrix $W_{ij} = e^{-\theta(L_j+A_{ij}-L_i)}$ where L_j denotes the smallest traveling time from j to the destination of the considered demand. The flow is assigned proportionally to the probability to use an arc before reaching the destination of the demand for the markov chain starting at the origin of the demand with a probability 1.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
// n is the number of nodes, m the number of arcs
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30;c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitMB(A,D,theta)
```

See Also : `LogitD` 28, `LogitB` 27, `LogitMD` 29, `LogitN` 30, `LogitNE` 31, `TrafficAssig` 43

7.18. **LogitMD** — Logit equilibrium (Markov Dial Method).

CALLING SEQUENCE :

```
[F]=LogitMB(A,D,theta)
```

PARAMETERS :

`A` : nxn nodesxnodes matrix of travel times
`D` : nxn nodesxnodes matrix of demand flows

`theta` : stochasticity parameter
`F` : nxn nodesxnodes assigned flow matrix

DESCRIPTION :
Compute the logit traffic assignment by the Markov method where on each road at each crossing we have to choose a link among the efficient links leaving this crossing excluding the ones coming back to the origin.

A Markov chain is defined by normalizing the following transition matrix $W_{ij} = e^{-\theta(L_j+A_{ij}-L_i)}$ where L_j denotes the smallest traveling time from j to the destination of the considered demand defined only on the efficient links. The flow is assigned

proportionally to the probability to use an arc before reaching the destination of the demand for the markov chain starting at the origine of the demand with a probability 1.

The graph must be strongly connected. If it is not the case we can add artificial arcs with large travel time in such a way that the new graph becomes strongly connected.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
// n is the number of nodes, m the number of arcs
n=10; m=40;
c1=m/(n*n);
A=sprand(n,n,c1)+diag(sparse(ones(n-1,1)),1);
A(n,1)=1;A=A-diag(diag(A));
// Demand generation, p is the number of demand.
p=30; c2=p/(n*n);D=sprand(n,n,c2);D=D-diag(diag(D));
//theta definition (almost determinsitic)
theta=10
// Flow calculation
FD=LogitMD(A,D,theta)
```

See Also : `LogitD` 28, `LogitB` 27, `LogitMB` 29, `LogitN` 30, `LogitNE` 31, `TrafficAssig` 43

7.19. **LogitN** Net Logit Assignment.

CALLING SEQUENCE :

```
LogitN(theta,method)
```

PARAMETERS :

`theta` : stochasticity parameter
`method` : macro among LogitB, LogitD, LogitMB, LogitMD
`%net` global variable NetList variable database

DESCRIPTION :
Compute the logit traffic assignment using a method among `LogitB, LogitD, LogitMB, LogitMD` the travel time is taken in `%net.links.time` and the assigned fow is put in the field `%net.links.flow` of the variable `%net` which is a NetList which must be declared global.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10;
// Flow calculation
LogitN(theta,LogitMB)
ShowNet()
```

See Also : LogitD 28, LogitB 27, LogitMB 29, LogitMD 29, LogitNE 31, TrafficAssig 43

7.20. **LogitNE** Net Logit Equilibrium.

CALLING SEQUENCE :

```
bench=LogitNE(theta,method,eps,Niter,Num)
```

PARAMETERS :

`theta` : stochasticity parameter
`method` : macro among LogitB, LogitD, LogitMB, LogitMD
`eps` : convergence error test
`Niter` : maximal number of iteration
`Num` : number of iteration already done
`bench` : niter x 5 matrix
`%net` : global NetList variable database

DESCRIPTION :
Compute the logit traffic equilibrium using a logit method among `LogitB, LogitD, LogitMB, LogitMD` the travel time is taken in `%net.links.time` and the assigned flow is put in the field `%net.links.flow` of the variable `%net` which is a `NetList` which must be declared global.

The equilibrium is computed by a divergent series method $F_{n+1} = F_n(1-\rho_n) + \rho_n f$ where f is the new assigned flow computed with the travelling time associated to the flow F_n, with $\rho_n = 1/(n+1)$.

If `%net.gp.verbose=%T` convergence informations are displayed. These informations are returned by the function in the variable bench.

The matrix `%net.gp.bench` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
// Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10
// Flow calculation
LogitNE(theta,LogitMD,1.e-6,30,0)
ShowNet()
```

See Also : LogitD 28, LogitB 27, LogitMB 29, LogitMD 29, LogitN 30, LogitNELS 32, TrafficAssig 43

7.21. LogitNELS — Net Logit Equilibrium (Linear Search).

CALLING SEQUENCE :

```
bench=LogitNELS(theta,method,eps,Niter)
```

PARAMETERS :

theta : stochasticity parameter
method : macro among LogitB, LogitD, LogitMB, LogitMD
eps : convergence error test
Niter : maximal number of iteration
bench : niter x 5 matrix
%net : global variable NetList
%VERBOSE : global boolean variable
% STACKSIZEREF : global integer variable

DESCRIPTION :

Compute the logit traffic equilibrium using a logit method among LogitB, LogitD, LogitMB, LogitMD the travel time is taken in %net.links.time and the assigned flow is put in the field %net.links.flow of the variable %net which is a NetList which must be declared global.

The equilibrium is computed using a linear search based on the corresponding deterministic criteria as approximated Lyapounov function. This method is valid at least for large theta.

If %net.gp.verbose=%T convergence informations are displayed. These informations are returned by the function in the variable %net.gp.bench.

EXAMPLES :

```
//  Graph generation (the graph must be stongly connected)
%net=TrafficExample('Steenbrink');
//theta definition (almost deterministic)
theta=10
// Flow calculation
LogitNELS(theta,LogitMD,1.e-6,40);
ShowNet()
```

See Also : LogitD 28, LogitB 27, LogitMB 29, LogitMD 29, LogitN 30, LogitNE 31, TrafficAssig 43

7.22. MSA — Method of successive averages.

CALLING SEQUENCE :

```
[f,ta,ben]=MSA(net,kmax,tol)
```

PARAMETERS :

net : a NetList
kmax : maximum number of iterations

`tol` : precision
`f` : assigned flow
`ta` : link travel time for the assigned flow `f`
`ben` : a `nix5` matrix (`ni` number of performed iterations) benchmark information

DESCRIPTION :
Assigns the flow with a MSA heuristic. After an initial `AON` assignment the links costs are updated and a new `AON` assignment is computed. The new flow, $f_{k+1} = (1-\rho_k)f_k+\rho_k y_k$, where $\rho_k = 1/k$, is computed as a combination of the previous flow f_k and an `AON` assignment y_k. This is repeated iteratively until iteration `kmax` or until the precision `tol` is reached.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Steenbrink');
[f,s,ben]=MSA(net,15,1e-3);
```

See Also : `AON` 18, `IAON` 25, `CAPRES` 21, `FW` 23, `TrafficAssig` 43

7.23. **MSASUE** — MSA algorithm for Stochastic User Equilibrium.

CALLING SEQUENCE :

```
[f,ta,ben]=MSASUE(net,beta,kmax,tol)
```

PARAMETERS :

`net` : a NetList
`beta` : variance of the perceived travel time over a road segment of unit travel time
`kmax` : maximum number of iterations
`tol` : precision
`f` : assigned flow (disaggregated by commodity)
`ta` : link travel time for the assigned flow f
`ben` : a `nix5` matrix (`ni` number of performed iterations) benchmark information

DESCRIPTION :
Assigns the flow with a MSA heuristic using a Probit-based model in each iteration. After an initial Probit assignment the links costs are updated and a new Probit assignment is computed using flow dependent travel times. The new flow, $f_{k+1} = (1-1/k)f_k+(1/k)y_k$, is computed as a combination of the previous flow f_k and a Probit assignment y_k. This is repeated iteratively until iteration `kmax` or until the precision `tol` is reached.

The matrix `ben` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
net=TrafficExample('Small');
[f,s,ben]=MSASUE(net,1,15,1e-3);
```

See Also : Probit 38, MSA32, FW 23, DSD 21, TrafficAssig 43

7.24. MakeNet — Makes a net list.

CALLING SEQUENCE :

```
net=MakeNet(nn,nx,ny,tl,hl,nf,lpp,td,hd,demand)
```

PARAMETERS :

nn : integer, the number of nodes in the net
nx : row vector, x coordinates of the nodes
ny : row vector, y coordinates of the nodes
tl : row vector, tail nodes numbers of the links
hl : row vector, head nodes numbers of the links
nf : integer between 0 and 6, travel time function formula
lpp : $4 \times n_l$ matrix, travel time function parameters, where n_l is the number of links
td : row vector, tail nodes numbers of the demands (its size is the number of commodities)
hd : row vector, head nodes numbers of the demands
demand : row vector, demand flows
net : a NetList

DESCRIPTION :

MakeNet makes a net list according to its arguments.

EXAMPLES :

```
// Coordinates for the nodes
nx=[500 10 500 900]
ny=[10 300 600 300]
// tail and head vectors for the links
tl=[1 2 3 1 1 3]
hl=[2 3 4 4 3 1]
// link-performance-function parameters
lpp=[1 1 1 1 1 1;1 3 5 2 7 1;1 2 1 2 1 2;2 2 2 2 2 2];
// Creation of the traffic net
%net=MakeNet(4,nx,ny,tl,hl,6,lpp,1,3,10);
// We can see it with
ShowNet()
```

See Also : NetList 35, ShowNet 42

7.25. Net2Par — Parameters from net.

CALLING SEQUENCE :

```
[g,d,lpp,nf]=Net2Par(net)
```

PARAMETERS :

`net` : a NetList
`g` : graph-list, graph of the network
`d` : 3 x nd matrix, tail, head and value of the demands
`lpp` : 4 x na matrix, data for the link performance functions
`nf` : integer between 0 and 6, travel time function formula

DESCRIPTION :
`Net2Par` recovers the data fields from a NetList.

EXAMPLES :

```
net=TrafficExample('Sioux Falls');
[g,d,lpp,nf]=Net2Par(net);
```

SEE ALSO : `IntroTrfAsg` 26, `NetList` 35, `Par2Net` 37, `TrafficExample` 45

7.26. NetList ——— Traffic Assignment Geographic Data Base.

NETLIST :
All the data needed and obtained in a traffic assignment problem are stored in a Scilab data structure that we call a NetList which is a typed list defined by :

```
tlist(['net','gp','nodes','links','demands'],gp,nodes,links,..
                                                        demands)
```

where :

- `gp` : contains the general properties of the net,
- `nodes` : contains the nodes parameters
- `links` : contains the links parameters
- `demands` : contains the demands parameters

Each one of those elements is itself a typed list.

GP :

- `node_number` : number of nodes in the network
- `link_number` : number of links in the network
- `demand_number` : number of demands (commodities)
- `lpf_model` : integer between 0 and 6 which defines the form of the link performance function.
- `verbose` (default `%T`): boolean when it is true information are given during assignment algorithm
- `algorithm` (default `'DSD'`): function name defining the algorithm choosen for the assignment
- `NodeDiameter` (default 30) : integer used in the graph window for displaying the network
- `NodeBorder` (default 1): integer giving thewidth of the circle border of the node symbols
- `FontSize` (default 10): integer used in the graph window
- `tolerance` (default 1e-6): real giving the precision used in algorithms
- `theta` (default 1): stochasticity parameter for Logit

- `Niter` (default 200): integer giving maximal number of iteration
- `N0` (default 0): integer giving starting number for Mean Average Algorithm
- `ShowDemands` (default `%T`): boolean when it is true the demands are shown by `ShowNet`
- `Show` (default `'flow'`): one of the string `'flow' 'cost' or 'time'`; it is used in `ShowNet` to choose the shown link weights
- `bench` (default 0): is an array storing the intermediary results computed by the algorithm when `gp.verbose=%T` this information is shown also during the computation.

NODES :

- `name` : string row of node names
- `x` : real row of the node x-coordinates
- `y` : real row of the node y-coordinates

LINKS :

- `name` : string row vector of link names
- `tail` : integer row vector of tail node numbers of the links
- `head` : integer row vector of head node numbers of the links
- `lpf_data` : array of 4 rows vectors giving the parameters of the lpf function of the links
- `flow` : real row vector of the total flows of the links (0 by default)
- `time` : real row vector of the time spent in the links
- `disaggregated_flow` : nd x na real matrix of the commodity flows on the links

DEMANDS :

- `name` : string row vector of demand names
- `tail` : integer row vector of tail node numbers
- `head` : integer row vector of head node numbers
- `demand` : real row vector of demand flows

EXTRACTING AND INSERTING DATA FROM A NETLIST :
As they are typed lists it is always possible to access to any field using the access functions, for example the x-coordinates of the nodes is `net1.nodes.x` .

LINK PERFORMANCE FUNCTION :

It is possible to choose among different formulas for the link performance functions with the value of `lpf_formula` field of `gp` denoted here `nf`.

$\texttt{nf} = 0$: $c = t_0 + m_a/(c_a F) + m_a[\max(0, F - c_a)]^{b_a}$, with $b_a \geq 1$

$\texttt{nf} = 1$: $c = t_0 e^{(F/ca)-1}$, with $c_a > 0$

$\texttt{nf} = 2$: $c = t_0 2^{(F/ca)-1}$, with $c_a > 0$

$\texttt{nf} = 3$: $c = t_0[1 + 0.15(F/c_a)^{m_a}]$, with $m_a \geq 1$ and $c_a > 0$

$\texttt{nf} = 4$: $c = t_0 + \log(c_a) - \log(c_a - F)$, with $c_a > 0$

$\texttt{nf} = 5$: $c = b_a - c_a(t_0 - b_a)/(F - c_a)$, with $c_a > 0$

$\texttt{nf} = 6$: $c = t_0 + c_a F + m_a F^{b_a}$, with $b_a > 1$

where the t_0, c_a, m_a, b_a are the row vectors of the matrix `net.links.lpf_data`

SHOWING THE NET :
When a net is shown, three types of arcs can be seen. The black, light-blue and cyan.

(1) The black and light-blue arcs correspond to the roads.
- The light blue to those roads with zero flow.
- The black to those with a non zero flow.

(2) The cyan arcs correspond to the OD-traffic demand. These arcs go from origin to destination.

See Also : `IntroTrfAsg` 26, `MakeNet` 34, `ShowNet` 42

7.27. **Par2Net** — Net from parameters.

CALLING SEQUENCE :

```
net=Par2Net(g,d,lpp,nf)
```

PARAMETERS :

`net` : a NetList
`g` : graph-list, graph of the network
`d` : 3xnd matrix, tail, head and value of demands
`lpp` : 4xna matrix, data for the link performance functions
`nf` : integer between 0 and 6, travel time function formula

DESCRIPTION :
`Par2Net` Assembles the data fields in a net.

EXAMPLES :

```
net=TrafficExample('Sioux Falls');
[g,d,lpp,nf]=Net2Par(net);
net2=Par2Net(g,d,lpp,nf)
```

See Also : `IntroTrfAsg` 26, `NetList` 35, `Net2Par` 34, `TrafficExample` 45

7.28. **Probit** — Probit-based stochastic network assignment.

CALLING SEQUENCE :

```
[f,s,ben]=Probit(net,ta,beta,accuracy,kmax)
```

PARAMETERS :

`net` : a NetList
`ta` : link travel time
`beta` : variance of the perceived travel time over a road segment of unit travel time
`accuracy` : precision
`kmax` : maximum number of iterations
`f` : assigned flow (disaggregated by commodity)
`s` : standard deviation for the assigned flow f

DESCRIPTION :
Assigns the flow with a Probit-based model. In this model the perceived travel time along any given path is normally distributed with mean `ta` and variance `beta*ta`. The algorithm use to make the assignment is base on a Monte Carlo simulation of the perceived link travel times.

EXAMPLES :

```
net=TrafficExample('Small');
ta=net.links.lpf_data(1,:);
// uses the example net Small
[f,s,ben]=Probit(net,ta,0.1,0.001,7);
```

See Also : `AON` 18, `IAON` 25, `CAPRES` 21, `FW` 23, `TrafficAssig` 43

7.29. **RandomNNet** — Random generation of traffic network data.

CALLING SEQUENCE :

```
net=RandomNNet(nn,mna,var,nd,nf)
```

PARAMETERS :

`nn` : number of nodes
`mna` : mean value of the exterior degree of each node (numbers of leaving arcs)
`var` : variance of the exterior degree of each node
`nd` : number of OD-pairs (origin-destination)
`nf` : model chosen for the computation of the travel time
`net` : resulting network

DESCRIPTION :
Generates randomly the data for the Traffic Assignment Problem. For each node an exterior degree is chosen randomly with a normal distribution of mean `mna` and variance `var`, then the successors are chosen without repetition. The value of `nf` indicates the formula chosen for the computation of

the travel time, by default it is 6. This way of generating randomly the net if very much slower than `RandomNet`.

EXAMPLES :

```
%net=RandomNNet(3,3,1,2);
// generates a network with 3 nodes,2 OD-pairs and form
// each node the number of leaving arcs is normally dis-
// tributed with mean 3 and var 1
ShowNet();
```

See Also : `IntroTrfAsg` 26, `RandomNet` 39, `Regular` 39, `NetList` 35, `ShowNet` 42

7.30. **RandomNet** — Random generation of traffic network data.

CALLING SEQUENCE :

```
net=RandomNet(nn,na,nd,nf)
```

PARAMETERS :

`nn` : number of nodes
`na` : number of arcs
`nd` : number of OD-pairs (origin-destination)
`nf` : model chosen for the computation of the travel time
`net` : the resulting Netlist

DESCRIPTION :
Generates randomly a strongly connected network with `nn` nodes, (approximately) `na` links and (approximately) `nd` OD pairs. The value of `nf` indicates the formula chosen for the computation of the travel time, by default it is 6.

EXAMPLES :

```
%net=RandomNet(3,5,2);
// generates a network with 3 nodes, 5 arcs and 2 OD-pairs
ShowNet();
```

See Also : `IntroTrfAsg` 26, `RandomNNet` 38, `Regular` 39, `ShowNet` 42

7.31. **Regular** — Generation of regular city traffic network data.

CALLING SEQUENCE :

```
net=Regular(hs,vs,nd)
```

PARAMETERS :

`hs` : integer number of horizontal streets
`vs` : integer number of vertical streets
`nd` : non-negative real parameter related to the OD-pairs (origin-destination)

`net` : the resulting Netlist

DESCRIPTION :
Generates a NetList consisting in a regular city whose graph is a `hs`×`vs` grid For the link travel time functions, the data is the same for all links and given by `t0=ca=ma, ba=1`. The `lpf_model` is 3. It is possible to generated 4 different types of Regular City depending on parameter `nd`:

- `nd=[]` : The net has all possible O-D pairs all of them with demand equal to 1
- `nd=1` : There is only one O-D pair joining the most distant nodes. Demand's value equal to 1.
- `nd` $\in (0,1)$: The parameter `nd` represent the density of O-D pairs present in the net. Demands value equal to 1.
- `nd>1` : There are `nd` random demands generated. Demands value in the interval $[0, 15]$.

EXAMPLES :

```
%net=Regular(4,5);
// generates a network with 4 horizontal streets
//  and 5 vertical streets,
// (20 nodes, 31 arcs) and all possible OD-pairs (343)
ShowNet();
%net=Regular(4,5,1);
// generates only one demand
ShowNet();
%net=Regular(4,5,0.5);
// generates randomly the OD-pairs with density 0.5
ShowNet();
%net=Regular(4,5,7);
// generates 7 random OD-pairs
ShowNet();
```

See Also : `IntroTrfAsg` 26, `TrafficExample` 45, `ShowNet` 42

7.32. **ShowDemands** Shows the demands of a net using Metanet or Scigraph.

CALLING SEQUENCE :

```
ShowDemands()
ShowDemands(nodes,demands,dmin,dmax)
```

PARAMETERS :

`%net` : the global variable NetList database
`nodes` : string or row vector of integers (nodes)
`demands` : string or row vector of integers (demands)
`dmin,dmax` : reals, minimum and maximum demands

DESCRIPTION :
`ShowDemands` shows the demands of `%net` that have volumes between `dmin` and `dmax`. All the demands are shown if nothing is indicated)

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
ShowDemands() // Shows the net
ShowDemands('all','between',100,200)
ShowDemands('used','between',100,200)
// Shows the net with the demands between 100 and 200,
// and the used nodes.
ShowDemands('used',[1 3 8 12],100,200)
// Shows the given demands with values between 100 and 200
```

See Also : ShowNet 42, ShowLinks 41, TrafficExample 45, TrafficAssig 43, IntroTrfAsg 26, NetList 35

7.33. **ShowLinks** — Shows the links of a net using Metanet or Scigraph.

CALLING SEQUENCE :

```
ShowLinks()
ShowLinks(nodes,arcs,fmin,fmax,tmin,tamx)
ShowLinks(nodes,arcs,fmin,fmax,tmin,tamx,d)
```

PARAMETERS :

`%net` : the global NetList varaiable database
`nodes` : string or row vector of integers (nodes)
`arcs` : string or row vector of integers (arcs)
`fmin,fmax` : reals, minimum and maximum flow
`tmin,tmax` : reals, minimum and maximum time
`d` : integer, a specific demand

DESCRIPTION :
Similar to `ShowNet` but it doesn't show the demands, it only shows the links of `%net` that verify the conditions given by the parameters (see `ShowNet`). If a demand is indicated, then the flow shown is the corresponding to that demand.

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
%net.gp.algorithm='AON';
TrafficAssig();
ShowLinks() // Shows the net
ShowLinks('used','between',3000,6000,0,0,3)
// Shows the net with the flow corresponding to demand 3,
// the light blue arcs are not used by the commodity 3
// but by other commodities. The width corresponds to
// the flow of the commodity 3. The arcs not shown have
// total flow less than 3000 or greater than 6000.
```

See Also : ShowNet 42, ShowDemands 40, TrafficAssig 43, IntroTrfAsg 26, NetList 35

7.34. **ShowNet** ——— Show a net using Metanet or Scigraph.
CALLING SEQUENCE :

```
ShowNet()
ShowNet(nodes,arcs,fmin,fmax,tmin,tamx,demands,dmin,dmax)
ShowNet(nodes,arcs,fmin,fmax,tmin,tamx,demands,dmin,dmax,d)
```

PARAMETERS :

`%net` : the global NetList varaible database
`nodes` : string or row vector of integers (nodes)
`arcs` : string or row vector of integers (arcs)
`fmin,fmax` : reals, minimum and maximum flow
`tmin,tmax` : reals, minimum and maximum time
`demands` : string or row vector of integers (demands)
`dmin,dmax` : reals, minimum and maximum demand
`d` : integer, a demand number

DESCRIPTION :
`ShowNet` shows the state of the `%net` with the flow and the travel time if they are already assigned. The conventions are that the non used arcs are light blue, the used arcs are black with their width proportional to the flow. The demands are magenta with their width proportional to the demand flow. The orientation of the demand arcs is from origin to destination.

It is possible to give a list of nodes, arcs and demands to be shown or a predefined string

- For the nodes the strings available are : `'all'` representing all the nodes, and `'used'` representing the nodes used by some route.
- For the arcs the options are `'all'` or `'between'` in this case the flow limits are the parameters `fmin, fmax` and the travel time limits are `tmin, tmax`.
- For the demands the options are `'all'` or `'between'` in this case the demand limits are `dmin,dmax`.

It is also possible to choose a demand in order to show the disaggregated flow corresponding to a specific commodity. In this case the limits on the flow given by fmin and fmax corresponds to the total flow but the width and the colors of the arcs correspond to the disaggregated flow.

EXAMPLES :

```
%net=TrafficExample('Steenbrink');
%net.gp.algorithm='DSDisaggregated';
TrafficAssig();
ShowNet() // Shows the net
ShowNet('used','between',1000,2000,0,0,3,0,10000,3)
// Shows the net with the flow corresponding to demand 3,
// the light blue arcs are not used by the commodity 3
// but by other commodities. The width corresponds to
// the flow of the commodity 3. The arcs not shown have
// total flow less than 1000 or greater than 2000.
```

See Also : `ShowLinks` 41, `ShowDemands` 40, `TrafficExample` 45, `TrafficAssig` 43, `IntroTrfAsg` 26, `NetList` 35

7.35. **TrafficAssig** — Traffic Assignment.

CALLING SEQUENCE :

```
TrafficAssig(p1,p2,p3)
```

PARAMETERS :

`%net` : : global NetList variable containing the Net database

`%net.gp.algorithm` : string to choose among `'AON'`, `'CAPRES'`, `'DSD'`, `'FW'`, `'IA'`, `'IAON'`, `'MSA'`, `'Probit'`, `'ProbitE'`, `'LogitB'`, `'LogitD'`, `'LogitMB'`, `'LogitMD'`, `'LogitBE'`, `'LogitDE'`, `'LogitMBE'`, `'LogitMDE'`

`p1,p2 p3` : different algorithms' parameters to change the default ones given in fields of `%net.gp` (see NetList).

DESCRIPTION :

Assigns the flow of the traffic network described by the NetList net using the algorithm selected by algo with the parameters given by p1, p2 and p3 (default in `%net.gp`) :

- `'AON'` : All or nothing
- `'CAPRES'` : Capres algorithm
- `'DSD'` : Disaggregated simplicial decomposition (only total flow is-computed). p1: iterations number, p2: precision
- `'DSDisaggregated'` : Disaggregated simplicial decomposition. p1: iterations number, p2: precision
- `'FW'` : Frank-Wolfe algorithm. p1: iterations number, p2: precision
- `'IA'` : Incremental assignment. p1: iterations number
- `'IAON'` : Iterated all or nothing. p1: iterations number
- `'MSA'` : Method of successive averages. p1: iterations number, p2: precision
- `'Probit'` : Probit assignment. p1: beta, p2: precision, p3: iterations number
- `'ProbitE'` : MSA algorithm for the stochastic user equilibrium (MSASUE). p1:beta,
  p2: precision, p3: iterations number
- `'LogitB'` : Logit assignment (all paths used not only efficient ones used)
- `'LogitD'` : Logit assignment (only efficient paths are used)
- `'LogitMB'` : Reiterated first step logit assignment (all paths used not only efficient ones used)
- `'LogitMD'` : Reiterated first step logit assignment (only efficient paths are used)
- `'LogitBE'` : Logit equilibrium assignment (all paths used not only efficient ones used)
- `'LogitDE'` : Logit equilibrium assignment (only efficient paths are used)
- `'LogitMBE'` : Reiterated first step logit equilibrium assignment (all-paths used

not only efficient ones used)

- `'LogitMDE'` : Reiterated first step logit equilibrium assignment (only efficient paths are used)
- `'NwtArc'` : Newton method bsed on nodes-links formulation (useful when there are a small number of demands)

The assigned flow, the current travel time and the disaggregated flow are stored in `%net`.

It is correct that for some parameters and network some assignment methods does not succeed to give an assignment. In particular, for Logit method, the effective domain of the stochasticity parameter is limited and depends of the network. For some networks and some methods this domain may be empty (for example LogitB with a zero travel time loop). For Logit method, the stochasticity parameter must not be too large to avoid numerical difficulties coming from the limited precision of the computation.

EXAMPLES :

```
//
// To load an editable traffic network graph
//
%net=TrafficExample('Steenbrink');
ShowNet();
//
// The black arcs are the roads and the cyan ones are the
// traffic OD demands. We can edit the net (see the net-edition
demo).
//
// To compute the traffic assignment with 'DSD' algorithm
//
TrafficAssig();
ShowNet();
//
// To show  the travel times select :
//       GRAPH/Display arc names/Time
//
// It is possible to assign the flow using another algorithm
// for example the incremental assignment denoted  'IA'
//
%net.gp.algorithm='IA'
%net.gp.Niter=50;
TrafficAssig();
ShowNet();
//
// For Logit assignment, the stochasticity parameter
// is in the field %net.gp.theta (default 1)
//
%net.gp.algorithm='LogitMD';
%net.gp.theta=3;
TrafficAssig();
ShowNet();
```

See Also : AON 18, CAPRES 21, DSD 21, FW 23, IA 24, IAON 25, MSA 32, Probit 38, LogitB 27, LogitD 28, LogitMB 29, LogitMD 29, LogitN 30, LogitNE 31, LogitNELS 32, MSASUE 33, IntroTrfAsg 26, NetList 35

7.36. TrafficExample — Traffic network example.

CALLING SEQUENCE :

```
net=TrafficExample('name',p1,p2,p3,p4)
```

PARAMETERS :

`net` : NetList of the traffic network
`name` : string to choose among `'Braess'`, `'Chicago'`, `'Diamond'`, `'Empty'`, `'Nguyen Dupuis'`, `'Normal Random City'`, `'Random City'`, `'Regular City'`, `'Sioux Falls'`, `'Small'`, `'Steenbrink'`, `'Triangle'`
`p1, p2, p3, p4` : reals needed for examples for `'Regular City'`, `'Random City'` and `'Normal Random City'`

DESCRIPTION :
Generates a net-list choosing among many traffic network examples:

- `'Braess'` : 4 nodes, 5 links and 1 demand. Linear travel time functions.
- `'Chicago'` : 546 nodes, 2176 links and 93135 demands. Real large example.
- `'Diamond'` : 8 nodes, 14 links and 1 demand, diamond shaped network.
- `'Empty'` : 2 nodes and 1 link graph.
- `'Nguyen Dupuis'` : 13 nodes, 19 links and 4 demands network. Linear travel time functions.
- `'Normal Random City'` : generated using `RandomNNet(p1,p2,p3,p4)`.
- `'Random City'` : generated using `RandomNet(p1,p2,p3,p4)`.
- `'Regular City'` : generated using `Regular(p1,p2,p3)`.
- `'Sioux Falls'` : 24 nodes, 76 links and 528 demands network. Travel time functions of power 4.
- `'Small'` : 4 nodes, 9 links and 2 demands network. Quadratic travel time functions.
- `'Steenbrink'` : 9 nodes, 36 links and 12 demands network. Linear travel time functions.
- `'Triangle'` : 3 nodes and 6 links network

These examples can be used as starting point to make different networks.

EXAMPLES :

```
%net=TrafficExample('Sioux Falls');
ShowNet()
```

See Also : RandomNet 39, RandomNNet 38, Regular 39, NetList 35, TrafficAssig 43

7.37. **WardropN** ——— Wardrop equilibrium (Newton hybrid method).

CALLING SEQUENCE :

```
bench=WardropN(a,k,eps,niter)
```

PARAMETERS :

`a` : regularizing coefficient (small number)
`k` : reducing regularizing coefficient exponential rate (number < 1)
`eps` : precision of the convergence test (small number)
`niter` : maximal number of iterations
`bench` : niter x 5 matrix
`%net` : global variable NetList

DESCRIPTION :
Compute the Wardrop equilibrium of a transport assignment problem by solving the following nodes-links variational formulation of the problem :

$$\min_q \sum_l C_l(f_l), \quad f_l = \sum_i q_{li}, \quad Hq_i = d_i, \quad 0 \leq q_{li}.$$

where :

- $C(f)$ is a cost in the classes defined by the lpf function,
- H is is the incidence nodes-arcs matrix of the network,
- q_{li} is the flow of the commodity i on the link l,

`%net` is the netlist containing all the information relative to the network and the cost function used.

The method used is a decomposed newton method in the space (q_i, v_i) $i = 1, p$, where the v_i denote the dual variables associated to the constraints $Hq_i = d_i$.

The variable a regularizes the matrices giving the potentials. At each iteration a is reduced by a factor k. We can try first a small a and $k = 1$.

This method is useful mainly in the case of a small number of commodities.

The matrix `bench` contains intermediary informations on the algorithm. For each iteration it gives the iteration number, time spent, cost, memory used and an convergence error evaluation.

EXAMPLES :

```
// Definition of the Network
nw=4;
%net=TrafficExample('Regular City',nw,nw);
// Traffic Assignment
bench=WardropN(0.1,2,1.e-4,12);
// Visualization of the NET
ShowNet();
```

See Also : `DSD` 21, `FW` 23, `Probit` 38, `MSASUE` 33, `LogitNE` 31, `LogitNELS` 32, `TrafficAssig` 43

7.38. dlpf — Plot the travel time functions.

CALLING SEQUENCE :

```
c=dlpf(f0,inc,f1,lpp,nf)
```

PARAMETERS :

`f0` : minimal flow
`inc` : increment
`f1` : maximal flow
`lpp` : matrix of travel time function parameters
`nf` : ttf formula
`c` : matrix of the computed costs

DESCRIPTION :
Plot the travel time functions for the given parameters (`lpp`) between the minimal flow `f0` and the maximal flow `f1` using a step of inc and the formula `nf`

EXAMPLES :

```
net=TrafficExample('Small');
[g,d,lpp]=Net2Par(net);
// To show the function with the different parameters
dlpf(0.1,.01,2,lpp,6);
```

See Also : `IntroTrfAsg` 26, `TrafficExample` 45, `lpf` 47

7.39. lpf — Travel time functions.

CALLING SEQUENCE :

```
[ta,dta,cta]=lpf(F,lpp,nf)
```

PARAMETERS :

`F` : row vector of flows in each arc
`lpp` : parameters of the travel time functions (ttf) for each arc
`nf` : model for the ttf
`ta` : travel time of each arc for the given flow
`dta` : derivate of the ttf for the given arc flow
`cta` : integral of the ttf for the given arc flow

DESCRIPTION :
Computes the travel time of each arc for the given flow using the travel time functions with parameters `lpp` and formula `nf`. Each colum of `lpp` is of the form `[t0; ca; ma; ba]` where the coefficients `t0` is the free-flow travel time and `ca` the practical capacity of the link. Associated to the `nf` number correspond a formula giving the travel time see `NetList` for the precise definition of these formulas.

EXAMPLES :

```
net=TrafficExample('Small');
[g,d,lpp,nf]=Net2Par(net);
F=rand(1,9);
[ta,dta,cta]=lpf(F,lpp,nf)
// shows the function with the different parameters
dlpf(0.1,.01,5,lpp,6);
```

See Also : IntroTrfAsg 26, TrafficExample 45, dlpf 47

References

[1] M.C. Bell "Alternativess to Dial's Logit Assignment Algorithm" Transp. Research, B, vol. 29B N. 4 pp. 287-295, 1995.

[2] M.G.H. Bell, Y. Iida "Transportation Network Analysis", J. Wiley & Sons, 1997.

[3] E. Cascetta "Transportation Systems Engineering : Theory and Methods", Kluwer Academic Press, 2001.

[4] R.B. Dial "A Probabilistic Multipath Traffic Assignment Model which Obviates Path Enumeration, transportation research, 5, 1971.

[5] C. Gomez (Editor) : "Engineering an Scientific Computing with Scilab", Birkhauser 1999 and (http://www-rocq.inria.fr/scilab/).

[6] T. Larsson, M. Patriksson :"Simplicial Decomposition with Disaggregated Representation for the Traffic Assignment Problem", Transportation Science, 26, 1992, 4-17.

[7] M. Patriksson : "The Traffic Assignment Problem, Models and Methods", VSP Utrecht 1994.

[8] Y. Sheffi "Urban Transportation Networks" Prentice Hall, Englewood Cliff, NJ, 1985.

[9] Toint, P. and Wynter, L. : "Asymmetric Multiclass Traffic Assignment: A coherent formulation", *Proceedings of the 13th International Symposium on Transportation and Traffic Theory*, ed. J.-B. Lesort, Pergamon, Exeter, UK., 1996.

[10] L. Wynter : "Contributions à la théorie et à l'application de l'affectation multiclasse du trafic", Thèse de doctorat de l'ENPC, novembre, 1995.

[11] L. Wynter : "A convergent algorithm for the multimodal traffic equilibrium problem", INRIA Research Report, to appear.

[12] S. Erlander, N.F. Stewart : "The Gravity Model in Transportation Analysis", VSP Utrecht 1990.

Index