Updating turn probabilities at intersections in response to traffic incidents*

Sylvie Thiébaux and Peter Lamb

CSIRO Mathematical & Information Sciences

PO Box 664

Canberra, ACT 2611

e-mail: <Firstname>.<Lastname>@cmis.csiro.au

tel: (61) 26 216 {7068, 7047}

fax: (61) 26 216 7111

Abstract

Urban road traffic simulators sometimes require the turn probabilities at the intersections of the network they are simulating. This report describes a method for updating the given probabilities following the occurrence of a traffic incident. Taking Dial's proportional assignment as the model of driver route choice behavior, we use a variant of Dial's algorithm to assign traffic to the turns of the network. We first estimate the origin and destination of traffic prior to the incident by computing an OD matrix (approximately) consistent with the assignment and the turn probabilities in normal traffic conditions. We then work backwards from the assignment in the changed conditions and the above OD matrix to determine a new set of turn probabilities appropriate in the new conditions. We validate this method on the urban networks constituting the test suite of the TRITRAM simulator.

*This Research was supported by the Cooperative Research Center for Advanced Computational Systems. Thanks to Garry Francis, Peter Lowrie, Ngoc Nguyen and Doug Quail from RTA Traffic Technology Branch, for their assistance in reviewing the paper and suggesting some of the evaluation procedures. Thanks also to Melissa Dobbie and Jeff Wood from CSIRO for their help with statistics.

1 Introduction

A traffic simulator like TRITRAM (TRaveller Information and TRAffic Management) [8] requires the turn probabilities at all intersections of the urban network under consideration. These probabilities and other traffic counts necessary for TRITRAM to operate have been gathered at several times reflecting typical traffic conditions over a 24 hours period.

Unfortunately, upon occurrence of an incident somewhere in the network, the given turn probabilities are unlikely to adequately model reality. In particular, they do not account for the change of route taken by drivers to avoid the congestion caused by the incident while still reaching their planned destination.

This report describes a method for automatically updating the turn probabilities following an incident. Briefly, the method works in two steps.

1. Based on the initial turn probabilities, we first estimate the origin and destination of traffic before the incident. This involves assuming a reasonable model of driver route choice behavior. That is, given the cost of travelling along each link of the network, and the origin and destination of a driver, we need to be able to assign the trip of that driver to the turns that he is probably going to make. The origin-destination (OD) matrix of the network can be determined from this assignment and the turn probabilities.

2. Following the incident, some of the link costs have been affected, which leads to a new assignment. In the second step then, we work backwards from this assignment posterior to the incident and from the OD matrix obtained at the previous step, to determine new turn probabilities consistent with these latter.

The report is organised as follows. Section 2 gives a more formal overview of the problem and the method. Section 3 describes the technical choices to be made in order to implement the method (choice of a link cost function, of an assignment model, of a numerical resolution method, etc ...). The following 3 sections detail the respective solutions chosen. Sections 7 and 8 gives experimental results obtained with the method on the networks constituting the test suite for TRITRAM. Section 9 summarises the benefits, limits and possible enhencements of the method.

2 Overview of the Method

2.1 Notations

Let $\mathcal{N}$ be the set of nodes of the network, $\mathcal{O} \subseteq \mathcal{N}$ be the set of origin nodes (input nodes of the network), $\mathcal{D} \subseteq \mathcal{N}$ be the set of destination nodes (sink nodes of the network), $\mathcal{P} = \mathcal{O} \times \mathcal{D}$ be the set of origin-destination pairs, $\mathcal{L}$ be the set of links, and $\mathcal{T} \subseteq \mathcal{L}^2$ be the set of valid turns (pairs of links).

We assume that we know the probability $\Pr(t)$ of making any turn $t \in \mathcal{T}$ while transiting trough the network. For $t = \langle l_1, l_2 \rangle$, this probability is the *joint* probabilities of travelling on link l_1 and travelling next on link l_2 during the transit. These probabilities are what we called the *turn probabilities* above. For any origin-destination pair $p = \langle o, d \rangle \in \mathcal{P}$, $\Pr(p)$ denotes the probability of making a trip from o to d. The set of such probabilities is called the *origin-destination* (OD) matrix. Further, for any turn $t \in \mathcal{T}$ $\Pr(t \mid p)$ is the probability that a trip from o to d makes turn t. The set of such probabilities is called the *assignment*.

2.2 Determining the OD Matrix

The first step of the method amounts to computing the OD matrix $\Pr(p)$ from the given turn probabilities $P(t)$ and the assignment $\Pr(t \mid p)$. Naturally, the probability of a turn is the sum of the probabilities of the trips between the origin-destination pairs making that turn. So we have:

$$\Pr(t) = \sum_{p \in \mathcal{P}} \Pr(t \mid p) \times \Pr(p)$$

These $|\mathcal{T}|$ equations can be gathered into a single matrix equation as follows.

- Let P be the $|\mathcal{T}|$-dimensional column vector representing the turn probabilities. That is, P_i is the probability of the i^{th} turn for $1 \leq i \leq |\mathcal{T}|$.
- Let X be the $|\mathcal{P}|$-dimensional column vector representing the OD matrix. That is, X_j is the probability of the j^{th} OD pair, for $1 \leq j \leq |\mathcal{P}|$.
- Let A be the $|\mathcal{T}| \times |\mathcal{P}|$ matrix representing the assignment. That is, $A_{i,j}$ is the assignment of the trips of the j^{th} OD pair to the i^{th} turn, for $1 \leq i \leq |\mathcal{T}|, 1 \leq j \leq |\mathcal{P}|$.

Then

$$P = AX \tag{1}$$

The OD matrix is the solution of this linear system of equations.

2.3 Updating the Turn Probabilities

In the second step, the cost of some of the links changes due to the incident. As a consequence, the assignment is changed, because drivers are attempting to avoid the congested areas of the network. Let A' be the new assignment. On the other hand, the OD matrix does not change, since drivers are still trying to reach their planned destination from their origin. Therefore, the new set of probabilities consistent with the OD matrix and the new assignment is P' such that

$$P' = A'X$$

A matrix multiplication is sufficient to retrieve the updated probabilities.

3 Technical Choices

There are a number of ways we can choose to implement this method, and a number of preliminary quantities to be computed from the available data. These available data consist in:

- the structure of the network (nodes, links, turns)
- the turn probabilities, given as *conditional* (not joint) link probabilities – that is, for turn $t = \langle l_1, l_2 \rangle$, we are given the probability of travelling next on link l_2 *given* that we are now travelling on link l_1.
- the flows on the input links of the network
- the number of lanes, speed limit, and length of each link.

3.1 Joint Probabilities, Link Costs, and Flows

A first preliminary step concerns the turn probabilities. The turn probability data available are typically conditional link probabilities rather than the joint link probabilities needed in matrix equation 1. Fortunately, the joint probabilities can easily be computed from the conditional probabilities and the flows on each input link of network. This amounts to solving a linear system of equations, which we detail in Section 4.

A first element to be chosen is the model of link cost. We chose the cost of the link to be defined as the travel time on that link. It is possible to get accurate travel time data computed by TRITRAM throughout the simulation. However, since our implementation is still experimental and not yet linked to TRITRAM, we have for the present opted for the transportation planning model used in the early development of NIMPAC [1], in which the travel time on a link depends on the flow, the speed limit, the length, and the number of lanes on that link. This model is detailed in section 4. Another could easily be substituted without affecting the rest of the implementation.

Since our cost model takes traffic flow into account, another preliminary step is to compute the flows on all links of the network. As will be seen in Section 4 this task is similar to that of computing the joint link probabilities from the conditional ones.

3.2 Assignment

The most important choice perhaps is that of the assignment model. Recall that assignment is the process of allocating traffic demand to the routes (and in particular the turns) of the network, in such a way as to model typical driver route choice behaviors.

Two main assignment categories exist in the literature: proportional and non-proportional assignments. With proportional assignments, it is usually assumed that the fraction of the traffic from a given origin to a given destination using a given route is proportional to the route's cost, and is independent of the traffic demand. Therefore, a proportional assignment can be determined independently of the OD matrix. Examples of proportional assignments range from all-or-nothing assignment, where the traffic is entirely assigned to a least-cost route from the origin to the destination, to multipath assignments where any reasonable route between the origin and destination can be taken, with a probability decreasing with the cost of this route.

With non-proportional assignments, link costs are generally assumed to be functions of the traffic demand. Thus, the re-partition of the traffic from a given origin to a given destination depends on the traffic demand for this OD pair and for the others. Consequently, the assignment cannot be determined without knowing the OD matrix. Matrix equation 1 tells us that, *vice versa*, the OD matrix cannot be determined without knowing the assignment. Therefore, the assignment and the OD matrix cannot be determined separately, and one has to solve for them simultaneously in equation 1. A typical example of this category is equilibrium assignment, where drivers travel along least-cost routes for an OD pair (Wardrop's first principle [11]), leading the traffic demand and re-partition to converge to an equilibrium.

Non-proportional assignment is usually better suited to congested networks. On the other hand, the computation of the OD matrix and the assignment is much more costly, and this is why we chose to use proportional assignment. Note however that our approach accounts to some extent for traffic congestion when computing the assignment, in that our link cost model is flow-dependent. In section 9, we will also discuss the possibility of extending the approach to simulate equilibrium assignment to some extent.

The assignment model we chose is Dial's multipath assignment [4]. The model and its implementation are detailed in Section 5. We refer to Dial's paper for a description of its advantages compared to other proportional assignment models, which, if one wished to, could be used interchangeably in our framework.

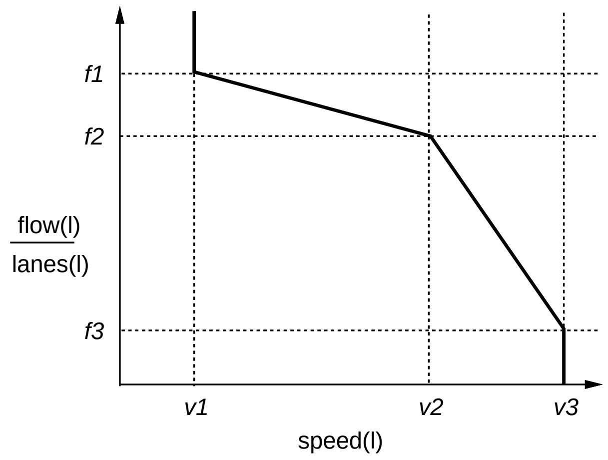
3.3 Solving the Matrix Equation

Once the assignment and the joint probabilities have been computed, one can determine the OD matrix from equation 1. However, this will rarely be as simple as solving an ordinary linear system of equations.

The first complication arises because there are usually more OD pairs than turn probabilities available. Therefore, the system is underdetermined and does not have a unique solution (there is potentially more than one OD matrix consistent with the assignment and the turn probabilities). A worse problem arises when the driver route choice behavior model assumed in the assignment is inconsistent with the available data (the turn probabilities). This happens when the model or/and the data do not reflect reality, and has the consequence that the matrix equation has no solution (there is no OD matrix consistent with both the turn probabilities and the assignment).

In order to overcome the first problem, one usually selects the solution maximising a certain objective function. Functions commonly used in the literature include least squares, two and infinite norms, and entropy [9]. As in [10, 7] we chose to select the OD matrix maximising entropy because this minimises the information implicitly assumed, in addition to the assignment and the turn probabilities, in order to obtain a unique solution. Any other choice would amount to further arbitrary assumptions of information, information which is not available to us.

Computing the solution with maximum entropy amounts to solving a non-linear system of equations. Several methods are available for that purpose, from very general ones like Newton's method, to very specialised ones such as MART (Multiplicative Algebraic Reconstruction Technique) [6] or SMART (Simultaneous MART) [2]. We chose to use SMART, because it provides a solution to the second problem mentioned above. When the system has no solution, the best that can be achieved is the computation of a useful approximation of the result. This is exactly what SMART does: it returns an OD matrix that is in some sense the closest to a solution.



$f_2 = 1800$ vehicles/h
$f_1 = 1.25\ f_2$
$f_3 = 0.1\ f_2$ if $\mathsf{lanes}(l) < 3$
and $0.2\ f_2$ otherwise

$v_1 = 8$ km/h
$v_2 = 40$ km/h
$v_3 = \mathsf{limit}(l)$

Figure 1: Average speed on a link as a function of the flow per lane

Section 6 details the entropy function, the SMART algorithm, and the approximation obtained in case of inconsistency between the assignment and the turn probabilities.

4 Link Costs, Flows, and Joint Probabilities

4.1 Link Costs

We start the detailed description of our technical choices with the cost model adopted in our current implementation. Recall that this model is expected to be replaced by TRITRAM data in the future.

Let $l \in \mathcal{L}$ be a link of length $\mathsf{length}(l)$, with a number of lanes $\mathsf{lanes}(l)$, a speed limit $\mathsf{limit}(l)$, and a traffic flow $\mathsf{flow}(l)$. We choose the *travel time* along link l to be the measure of that link's cost. So if we denote $\mathsf{speed}(l)$ the average speed of the traffic on l, then the cost of l is:

$$\mathsf{cost}(l) = \frac{\mathsf{length}(l)}{\mathsf{speed}(l)} \tag{2}$$

We compute the average speed from the flow, number of lanes and speed limit, based on the model used in the early development of NIMPAC [1]. This model is given in Figure 1, from which we see that:

$$\mathsf{speed}(l) = \begin{cases} v_1 & \text{if } \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} \geq f_1 \\ \frac{v_1 - v_2}{f_1 - f_2} \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} + \frac{v_2 f_1 - v_1 f_2}{f_1 - f_2} & \text{if } f_2 \leq \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} < f_1 \\ \frac{v_2 - v_3}{f_2 - f_3} \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} + \frac{v_3 f_2 - v_2 f_3}{f_2 - f_3} & \text{if } f_3 \leq \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} < f_2 \\ v_3 & \text{if } \frac{\mathsf{flow}(l)}{\mathsf{lanes}(l)} < f_3 \end{cases} \tag{3}$$

4.2 Link Flows

Our cost model requires the flows on all the network links to be computed beforehand. This can easily be done from the available data, namely the from flows on the input links and the conditional link probabilities. Let us then note $\mathrm{Tr}(l', l)$ the conditional probability of travelling next on link $l \in \mathcal{L}$ given that we are now travelling on link $l' \in \mathcal{L}$. If l is not an input link of the network, the flow on l comes from its incoming links in the proportion indicated by Tr, that is:

$$\mathsf{flow}(l) = \sum_{l' \in \mathcal{L}} \mathrm{Tr}(l', l)\ \mathsf{flow}(l')$$

If l is an input link, then we already know the flow on l.

We can write these equations in the following form, were the input links of the network have been given lower indices:

$$\begin{pmatrix} \mathsf{flow}(1) \\ \mathsf{flow}(2) \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\mathrm{Tr}(1, l) & -\mathrm{Tr}(2, l) & \cdots & 1 - \mathrm{Tr}(l, l) & \cdots & -\mathrm{Tr}(|\mathcal{L}|, l) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\mathrm{Tr}(1, |\mathcal{L}|) & -\mathrm{Tr}(2, |\mathcal{L}|) & \cdots & -\mathrm{Tr}(l, |\mathcal{L}|) & \cdots & 1 - \mathrm{Tr}(|\mathcal{L}|, |\mathcal{L}|) \end{pmatrix} \begin{pmatrix} \mathsf{flow}(1) \\ \mathsf{flow}(2) \\ \vdots \\ \mathsf{flow}(l) \\ \vdots \\ \mathsf{flow}(|\mathcal{L}|) \end{pmatrix}$$

That is, if we note:

- K the $|\mathcal{L}|$-dimensional column vector such that $K_i = 0$ if the i^{th} link is not an input link, and equals the known value that link's flow otherwise,
- T the $|\mathcal{L}| \times |\mathcal{L}|$ matrix such that $T_{ij} = \mathrm{Tr}(j^{th} \text{ link}, i^{th} \text{ link})$,
- I the $|\mathcal{L}| \times |\mathcal{L}|$ identity matrix,

then, the vector F such that F_i is the flow on the i^{th} link is the solution of the matrix equation:

$$K = (I - T)\ F$$

It suffices to solve this linear system of equations to obtain the all flows, and to report them in Equations 3 and 2 to get the the link speeds, and the link costs. The system will always have a unique solution (unless Tr is not a conditional probability distribution!).

4.3 Joint Link Probabilities

Our approach requires the values of the joint link probabilities for the network, since they underlie matrix equation 1. The computation of these is trivial once the link flows are available. Indeed, for any turn $t = \langle l_1, l_2 \rangle \in \mathcal{T}$, the conditional probability formula is:

$$\mathrm{Tr}(l_1, l_2) = \frac{\mathrm{Pr}(\langle l_1, l_2 \rangle)}{\mathrm{Pr}(l_1)}$$

The probability $\mathrm{Pr}(l_1)$ of travelling on link l_1 is simply the fraction of the input traffic flow travelling on that link. So, letting $\mathcal{I} \subseteq \mathcal{L}$ denote the set of input links of the network, we have:

$$\mathrm{Pr}(\langle l_1, l_2 \rangle) = \mathrm{Tr}(l_1, l_2) \frac{\mathsf{flow}(l_1)}{\sum\limits_{l \in \mathcal{I}} \mathsf{flow}(l)}$$

4.4 An Example

Let us now take the example of the network of the Oxford Street area in Sydney. The network is shown in Figure 2. The figure indicates the labels of the links and of the origin/destination nodes.

Tables 1 and 2 give the available data in plain (non-bold) digits. These are:

- the links together with their number of lanes, length, speed limit, and known flows for the input links,
- the turns characterised by their source link, destination link, and conditional link probability. Note that in order to make the example more interesting for the remainder of this report, we changed the original flow of 1044 on input link 330 to 3134.

In the tables, the results are given in bold digits. These are:

- the flows for the non-input links of the network, the link costs,
- the joint link probabilities for each turn.

Let us examine link 420. The flow on that link come from links 320 and 330, so:

$$\begin{aligned}
\mathsf{flow}(420) &= \mathsf{flow}(330) \times \mathrm{Pr}(\langle 330, 420 \rangle) + \mathsf{flow}(320) \times \mathrm{Pr}(\langle 320, 420 \rangle) \\
&= 3134 \times 0.4 + 596 \times 0.9 \\
&= 1790
\end{aligned}$$

link l	lanes(l)	length(l)	limit(l)	flow(l)	cost(l)
1010	3	400	60	**1139.5**	**24.11**
1110	3	400	60	**2245.8**	**26.37**
1210	2	400	60	**474.3**	**24.29**
1310	3	400	60	**911.7**	**24.00**
1410	3	400	60	**1101.7**	**24.04**
1510	3	400	60	**1935**	**25.70**
1610	4	400	60	**572.9**	**24.00**
110	4	130	60	**2444.6**	**8.28**
120	2	400	60	538	**24.45**
210	4	400	60	2200	**25.1**
130	3	170	60	**1283.4**	**10.36**
220	3	130	60	**2164.2**	**8.51**
310	3	170	60	**1356.8**	**10.42**
140	3	115	60	**1355.1**	**7.05**
230	4	150	60	**326.2**	**9.00**
320	3	400	60	596	**24.00**
410	3	115	60	**1625.8**	**7.20**
330	4	400	60	3134	**26.61**
420	4	110	60	**1790**	**6.74**
510	4	150	60	**650**	**9.00**
430	3	400	60	1582	**24.97**
520	3	230	60	**1707.7**	**14.50**
610	4	400	60	262	**24.00**
710	4	400	60	668	**24.00**
620	4	185	60	**1185.7**	**11.10**
810	4	160	60	**1784.8**	**9.80**
720	4	155	60	**308.2**	**9.30**
910	4	400	60	**1619.07**	**24.25**
820	4	400	60	1020	**24.00**

Table 1: Link input data (plain) and results (bold face)

l_1	l_2	$\mathrm{Tr}(l_1, l_2)$ (%)	$\mathrm{Pr}(\langle l_1, l_2 \rangle)$ (%)
120	310	25	**1.35**
120	410	75	**4.04**
220	1110	100	**21.64**
140	220	65	**8.81**
320	1310	5	**0.30**
420	1510	20	**3.58**
140	1210	35	**4.74**
320	420	90	**5.36**
520	1610	5	**0.85**
420	520	75	**13.43**
320	130	5	**0.30**
620	720	5	**0.59**
520	810	95	**16.22**
420	140	5	**0.90**
720	510	5	**0.15**
620	1010	95	**11.26**
820	910	10	**1.02**
720	230	95	**2.93**
820	620	90	**9.18**
110	310	50	**12.22**
210	110	100	**22.00**
110	410	50	**12.22**
310	1410	35	**4.75**
130	220	100	**12.83**
310	1310	65	**8.82**
230	1110	25	**0.82**
410	520	3	**0.49**
330	420	40	**12.54**
230	110	75	**2.45**
410	1510	97	**15.77**
510	810	25	**1.63**
330	130	40	**12.54**
430	520	20	**3.16**
510	1610	75	**4.88**
610	1010	5	**0.13**
710	230	5	**0.33**
330	1410	20	**6.27**
430	140	80	**12.66**
610	720	95	**2.49**
810	620	15	**2.68**
710	510	95	**6.35**
810	910	85	**15.17**

Table 2: Turn input data (plain) and results (bold face)

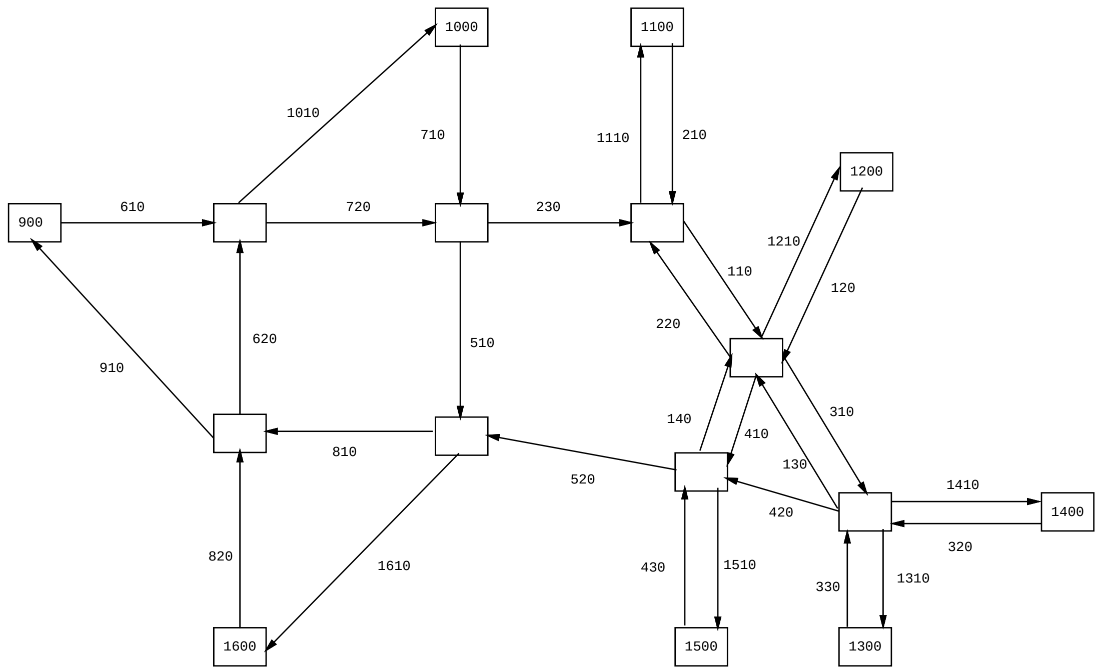


Figure 2: The Oxford Street network

Now the average speed on link 420 is:

$$
\begin{aligned}
\mathsf{speed}(420) &= \frac{v_2 - v_3}{f_2 - f_3}\frac{\mathsf{flow}(420)}{\mathsf{lanes}(420)} + \frac{v_3 f_2 - v_2 f_3}{f_2 - f_3} \\
&= \frac{40-60}{1800-360}\frac{1790}{4} + \frac{60\times 1800 - 40\times 360}{1800-360} \\
&= 58.78\mathrm{km/h}
\end{aligned}
$$

which yield a cost of:

$$
\begin{aligned}
\mathsf{cost}(420) &= \frac{\mathsf{length}(420)}{\mathsf{speed}(420)} \\
&= \frac{110\times 3600}{1000\times 58.78} \\
&= 6.74
\end{aligned}
$$

Last let us compute the probability of making a turn from link 420 to link 140:

$$
\begin{aligned}
\Pr(\langle 420, 140\rangle) &= \frac{\mathrm{Tr}(420,140)\times\mathsf{flow}(420)}{\mathsf{flow}(120)+\mathsf{flow}(210)+\mathsf{flow}(320)+\mathsf{flow}(330)+\mathsf{flow}(430)+\mathsf{flow}(610)+\mathsf{flow}(710)+\mathsf{flow}(820)} \\
&= \frac{0.05\times 1790}{538+2200+596+3134+1582+262+668+1020} \\
&= 0.895\%
\end{aligned}
$$

5 Dial's Assignment

We now detail the principles behind Dial's assignment, and give a variant of Dial's algorithm assigning traffic from a given origin to a given destination to the turns of the network[1].

5.1 Assumptions and Requirements

Dial's assignment is a proportional multipath assignment, meaning that the fraction of the traffic from a given origin to a given destination assigned to a given route is proportional to that route's cost, and that more than one route may be assigned some of the traffic. Dial's assignment makes the following assumption about driver route choice behavior:

1. Drivers may use all *reasonable* routes from the origin to the destination with a non-zero probability. They will not use unreasonable routes, that is, routes that contains turns that take the driver closer to his origin or further away from his destination.

2. Drivers take all reasonable routes of equal cost with the same probability.

3. When faced with two reasonable routes of unequal cost, drivers take the cheapest with a higher probability.

Two further requirements are imposed on the algorithm implementing these principles:

4. In order to be flexible, it should provide means of controlling the route diversion probabilities.

5. In order to be efficient, the algorithm should not explicitly enumerate routes.

[1] Dial's original assignment algorithm assigns traffic to links rather than turns. It is meant to operate in conjunction with a method for estimating OD matrices from the flows on the network's links. The variant we present here is based on turns, since the turn probabilities are available to us, and that these are more informative than the link flows. Note however that it is possible to simulate our variant with the original algorithm by introducing new nodes and links in the network. For each internal link l, two additional nodes $n_{l,i}$ and $n_{l,o}$ should be created, one at each extremity of l (for network input links, only create $n_{l,o}$ and for output links $n_{l,i}$). For each turn $t = \langle l_1, l_2 \rangle$, one additional link l_t should be created, linking $n_{l_1,o}$ to $n_{l_2,i}$. The original turn probabilities can be recast as flows on the new links.

5.2 Principles

5.2.1 Route Probability

We start by defining the probability that a particular route is taken when travelling from the origin to the destination. Let $o \in \mathcal{O}$ be the origin node and $d \in \mathcal{D}$ be the destination. Let c^* be the cost of the cheapest route from o to d. The probability of taking a reasonable route of cost c linking o to d is defined to be *proportional* to:

$$\exp \theta(c^* - c) \tag{4}$$

for some user-supplied parameter $\theta \in \mathbb{R}^+$. The probability of taking any unreasonable route is defined to be 0.

This trivially complies with requirement 1. This complies with requirement 2 because two routes with identical cost c have the same probability. This complies with requirement 3 because if the cost c_1 of a route is higher than the cost c_2 of another, then the probability proportional to $\exp \theta(c^* - c_1)$ assigned to the former route is lower than the probability proportional to $\exp \theta(c^* - c_2)$ assigned to the latter. This also complies with requirement 4 because θ can be used to control the diversion probabilities. If $\theta = 0$, then all reasonable routes have the same probability of being taken. When θ increases, the probability of using a cheaper route increases. For large θ, the assignment asymptotically approximates all-or-nothing.

One of the key features of Dial's algorithm is that it is able to assign traffic to all routes according to the probabilities given above, without having to enumerate these routes in any way. The first trick to achieve this is not to work directly with the route probabilities, but rather with the likelihood of the turns long these routes.

5.2.2 Turn Likelihood

We define the likelihood of each turn $t = \langle l_1, l_2 \rangle$ as the exponential (times θ) of the difference of cost incurred by going from o to l_2 via l_1 compared to that of the least-cost route from o to l_2. The interest of such a definition, is that the route probability as defined above is proportional to the product of the likelihoods of the turns along this route.

More formally, let $p(l)$ denote the cost of the cheapest route from the origin o to (the start of) link $l \in \mathcal{L}$, and $q(l)$ the cost of the cheapest route from (the start of) l to the destination d. t is part of a reasonable route from o to l_2 iff $p(l_1) < p(l_2)$ and $q(l_2) < q(l_1)$. If this is not the case, taking t leads us to backtrack towards o, or to get further away from d.

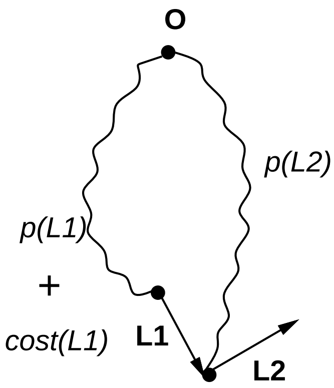


Figure 3: Cost incurred by taking a turn

Now, the cost of the cheapest route from o to l_2 is $p(l_2)$, while the cheapest route going from o to l_2 via l_1 has cost $p(l_1) + \mathsf{cost}(l_1)$. Therefore, the difference of cost incurred by taking t is $p(l_2) - p(l_1) - \mathsf{cost}(l_1)$. This is illustrated in Figure 3. Summing up, the likelihood $h(t)$ of turn t is:

$$\begin{cases} \exp\theta(p(l_2) - p(l_1) - \mathsf{cost}(l_1)) & \text{if } p(l_1) < p(l_2) \text{ and } q(l_2) < q(l_1) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

It can easily be verified that the product of the $h(t)$ along a route is 0 if this route is unreasonable, and equals the quantity given in (4) otherwise.

5.2.3 Turn Probabilities

The core of the assignment algorithm computes the probability of each turn (joint probabilities of its links) from the turn likelihoods. For that purpose, let us define $b(l)$ to be proportional to the sum of the probabilities of the routes leading from o to the end of link $l \in \mathcal{L}$. Reciprocally, we define $a(l)$ to be proportional to the sum of the probabilities of the routes leading from the start of l to d. Figure 4 represents $b(l_1)$ and $a(l_2)$ for turn $t = \langle l_1, l_2 \rangle \in \mathcal{T}$.

The probability of turn t is the sum of the probabilities of all routes taking that turn. These routes are those represented in Figure 4. Since the probability of a route is proportional to the product of the likelihoods of the turns along this root, then the probability of turn t is proportional to:

$$b(l_1) \times h(\langle l_1, l_2 \rangle) \times a(l_2) \tag{6}$$

Again, this is illustrated in Figure 4.

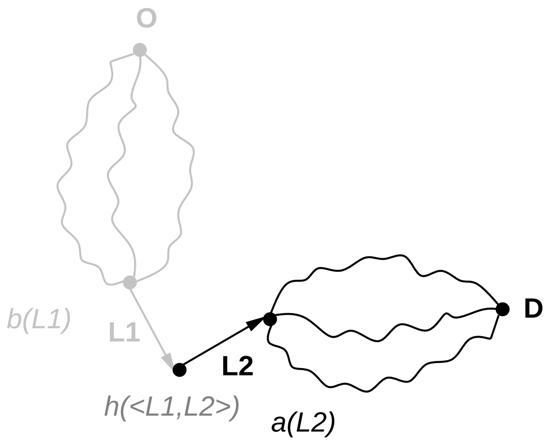


Figure 4: All routes from o to d going through turn $t = \langle l_1, l_2 \rangle$

Note that by definition of the function a, the sum of the probabilities of all routes from o to d is proportional to:

$$\sum_{l \in \mathsf{out}(o)} a(l)$$

where $\mathsf{out}(o)$ is the set of links starting at the origin o. Since the probabilities must sum to 1, then the proportionality coefficient is $K = \frac{1}{\sum_{l \in \mathsf{out}(o)} a(l)}$.

5.3 The Algorithm

The algorithm has to perform the following steps:

1. based on the cost of a link, compute the cost of the cheapest routes from each link $l_1 \in \mathcal{L}$ to each link $l_2 \in \mathcal{L}$
2. for each origin $o \in \mathcal{O}$ (resp. each destination $d \in \mathcal{D}$), compute $p(l)$ wrt. o (resp. $q(l)$ wrt. d) for each link $l \in \mathcal{L}$
3. for each OD pair $\langle o, d \rangle \in \mathcal{P}$
   (a) use p and q to compute the likelihood $h(t)$ of each turn $t \in \mathcal{T}$
   (b) use h to compute $b(l)$ and $a(l)$ for each link $l \in \mathcal{L}$, and then K
   (c) use b, a, h and K to compute the probability of each turn $t \in \mathcal{T}$
   (d) report the result in the assignment matrix A

The simplest way of performing step 1. is to use the Floyd-Warshall algorithm on the graph formed by the links of the network, rather than on the original graph

inputs:
  the set of links $\mathcal{L}$
  the set of turns $\mathcal{T}$
  the link cost function cost

output:
  the $\mathcal{L} \times \mathcal{L}$ matrix M such that $M[l_1, l_2]$ is the cost
  of the cheapest route between links l_1 and l_2

/* initialisation of M */
for $l_1 \in \mathcal{L}$ **do**
  for $l_2 \in \mathcal{L}$ **do**

$$M[l_1, l_2] \leftarrow \begin{cases} 0 & \text{if } l_1 = l_2 \\ \mathsf{cost}(l_1) & \text{if } \langle l_1, l_2 \rangle \in \mathcal{T} \\ \infty & \text{otherwise} \end{cases}$$

  od
od

/* transitive Closure */
for $l_3 \in \mathcal{L}$ **do**
  for $l_1 \in \mathcal{L}$ **do**
    if $M[l_1, l_3] < \infty$ **then**
      for $l_2 \in \mathcal{L}$ **do**
        $M[l_1, l_2] \leftarrow \mathsf{min}(M[l_1, l_2], M[l_1, l_3] + M[l_3, l_2])$
        od
    fi
  od
od

Figure 5: The Floyd-Warshall algorithm applied to the link graph.

formed by the nodes of the network. The resulting algorithm is given in Figure 5. Another alternative which exhibits better asymptotic run-time performance, but is more complex to implement is Dijkstra's algorithm [5].

For step 2, note that the p vector (resp. q vector) relative to a given origin o (resp. to a given destination d) can be computed from the cheapest routes matrix M by taking the minimum of the $M[x, l]$ over the links x starting from o (resp. of the $M[l, x] + \mathsf{cost}(x)$ over the links x arriving at d).

Step 3a. simply amounts to reporting the appropriate values in equation 5.

Step 3b. is trickier. We have to compute the sum of the probabilities of the routes from one point to another without enumerating these routes. Fortunately, this can be done using a dynamic programming algorithm. For computing b, we start by noticing that for all links l having o as origin, the sum of the probabilities of the routes from o to l's origin is 1. That is: $b(l) = 1$ for these links. Now, if we take the other links l in ascending order with respect to their distance $p(l)$ from the origin o, then the sum of the probabilities of the routes from o to the

inputs:
    the set of links $\mathcal{L}$
    the set of turns $\mathcal{T}$
    the origin $o \in \mathcal{O}$ and the destination $d \in \mathcal{D}$
    the least cost vectors p and q relative to $\langle o, d \rangle$
    the turn likelihood vector h relative to $\langle o, d \rangle$

outputs:
    the $\mathcal{L}$-dimensional vector b (resp. a) such that $b(l)$ (resp. $a(l)$) is proportional to the sum of the probabilities of the routes from o to l's destination (resp. from l's origin to d)

/* base case for b */
for $l \in \mathcal{L}$ **do**
    if $o = \mathsf{origin}(l)$ **then**
        $b[l] \rightarrow 1$
    else $b[l] \rightarrow 0$
    fi
od

/* recursion step for b*/
for $l_2 \in \mathcal{L}$ **in-increasing-order-of** p **do**
    if $p[l_2] < \infty$ **and** $b[l_2] = 0$ **then**
        for $l_1 \in \mathcal{L}$ **such that** $\langle l_1, l_2 \rangle \in \mathcal{T}$ **do**
            $b[l_2] \leftarrow b[l_2] + h[\langle l_1, l_2 \rangle] \times b[l_1]$
        od
    fi
od

/* base case for a */
$\alpha \leftarrow \min_{l \text{ s.t. } d = \mathsf{destination}(l)} p(l) + \mathsf{cost}(l)$
for $l \in \mathcal{L}$ **do**
    if $d = \mathsf{destination}(l)$ **and** $p(l) < \alpha$ **and** $q(l) > 0$ **then**
        $a[l] \rightarrow \exp \theta(\alpha - p(l) - \mathsf{cost}(l))$
    else $a[l] \rightarrow 0$
    fi
od

/* recursion step for a*/
for $l_1 \in \mathcal{L}$ **in-increasing-order-of** q **do**
    if $q[l_1] < \infty$ **and** $a[l_1] = 0$ **then**
        for $l_2 \in \mathcal{L}$ **such that** $\langle l_1, l_2 \rangle \in \mathcal{T}$ **do**
            $a[l_1] \leftarrow a[l_1] + h[\langle l_1, l_2 \rangle] \times a[l_2]$
        od
    fi
od

Figure 6: Computation of a and b

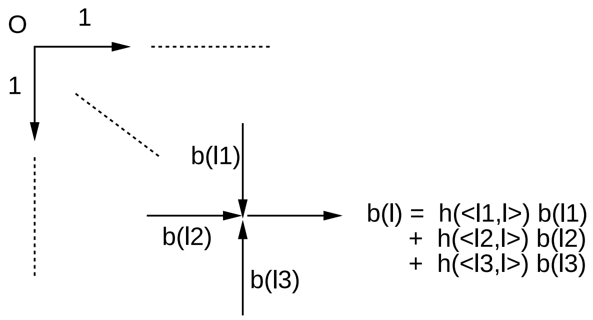


Figure 7: Computation of b

origin of l only depends on the sum of the probabilities of the routes from o to the origin of the links l' turning onto l, and on the likelihoods of the respective turns: That is:

$$b(l) = \sum_{l' \text{ s.t. } \langle l', l\rangle \in \mathcal{T}} h(\langle l', l\rangle)\ b(l')$$

This is illustrated in Figure 7. Computing a is very similar. The algorithm for this step is given in Figure 6.

Step 3c. simply consists in reporting the adequate values in equation 6 and multiplying by K.

Step 3d. consists in incorporating into the appropriate column of assignment matrix A, the probabilities found for each turn t for the OD pair $\langle o, d\rangle$ currently under consideration (recall that the rows of the assignment matrix A represent the turns, while the columns represent the OD pairs). The matrix being sparse (a given turn is only taken to travel between a few OD pairs), it is best to encode it as a vector, giving, for each turn t, the set of pairs (p, α) such that t is taken with probability $\alpha > 0$ when travelling from o to d.

The complexity of the steps are as follows:

step 1.	$O(\lvert\mathcal{L}\rvert^3)$
step 2.	$O((\lvert\mathcal{O}\rvert + \lvert\mathcal{D}\rvert)\lvert\mathcal{L}\rvert)$
step 3a.	$O(\lvert\mathcal{T}\rvert)$
step 3b.	$O(\lvert\mathcal{L}\rvert \log(\lvert\mathcal{L}\rvert))$
step 3c.	$O(\lvert\mathcal{T}\rvert)$
step 3d.	$O(\lvert\mathcal{T}\rvert)$

Since the number of turns is of the same order as the number of links, this yields a total complexity of:

$$O(|\mathcal{L}|^3 + |\mathcal{O}||\mathcal{D}||\mathcal{L}| \log(|\mathcal{L}|))$$

Note that $|\mathcal{O}|$ and $|\mathcal{D}|$ are typically much smaller than $|\mathcal{L}|$, so the dominant complexity for large networks is that of finding the shortest paths matrix. However,

l_1	l_2	$\Pr(\langle l_1, l_2\rangle \mid \langle 1300, 1100\rangle)$ (%)
330	420	3.2
330	130	96.8
420	140	3.2
130	220	96.8
220	1110	100

Table 3: Assignment from origin 1300 to destination 1100

the figure is different if Dijkstra's algorithm is used in place of Floyd-Warshall. Indeed, Dijkstra can compute the costs we need (rather than the whole all pairs shortest paths matrix) in $O((|\mathcal{O}| + |\mathcal{D}|)|\mathcal{L}| \log(|\mathcal{L}|)$. Since $|\mathcal{O}| + |\mathcal{D}| << |\mathcal{O}||\mathcal{D}|$, the dominant complexity is that of the dynamic programming algorithm in step 2b.

5.4 An Example

As an example, let us fill the column of the assignment matrix corresponding to OD pair $\langle 1300, 1100\rangle$ for the Oxford Street network.

Table 4 gives the cost of each link l (taken from Table 1), and the cost $p(l)$ of the least-cost path from the origin 1300 to the start of l as well as the cost $q(l)$ of the least-cost path from the start of l to the destination d, computed using the Floyd-Warshall algorithm in Figure 5.

From these and equation 5, we can compute the turn likelihood values given in the top graphic of Figure 8. A value of $\theta = 1$ was used. In the figure, the turns having a non-zero likelihood (the reasonable turns) are represented with dashed lines and their likelihood is indicated. Note that all reasonable turns have a likelihood of 1 except the turn from link 140 to link 220 (represented with bold lines) which has a likelihood of 0.033.

The two remaining graphics in Figure 8 give the non-zero values of b and a, computed by the algorithm in Figure 6. Note that for computing b for instance, a possible order for examining the links is 330, 130, 140, 220, 1110.

Table 3 gives the resulting assignment column (non-zero probabilities only). Note that most of the traffic goes via link 130, rather than via links 420 and 140, because this is the least cost route. Note also that the proportionality coefficient K equals 1/1.033. So, for instance, the probability of taking the turn from link 420 to link 140 when traveling from the origin 1300 to the destination 1100 is:

$$\begin{aligned} \Pr(\langle 420, 140\rangle \mid \langle 1300, 1100\rangle) &= \frac{b(420) \times h(\langle 420, 140\rangle) \times a(140)}{a(1100)} \\ &= \frac{1 \times 1 \times 0.033}{1.033} \\ &= 3.2\% \end{aligned}$$

link l	cost(l)	p(l)	q(l)
1010	24.11	68.74	∞
1110	26.37	45.48	26.37
1210	24.29	40.39	∞
1310	24.00	105.75	∞
1410	24.04	26.61	∞
1510	25.70	33.34	∞
1610	24.00	47.84	∞
110	8.28	87.04	95.55
120	24.45	∞	111.72
210	25.10	∞	120.66
130	10.36	26.61	45.25
220	8.51	36.97	34.89
310	10.42	95.32	∞
140	7.05	33.34	41.94
230	9.00	78.04	35.37
320	24.00	∞	69.25
410	7.20	95.32	87.27
330	26.61	0	71.86
420	6.74	26.61	48.67
510	9.00	78.04	74.57
430	24.97	∞	66.90
520	14.50	33.34	80.07
610	24.00	∞	68.67
710	24.00	∞	59.37
620	11.10	57.64	55.77
810	9.80	47.85	65.57
720	9.30	68.74	44.67
910	24.25	57.64	∞
820	24.00	∞	79.77

Table 4: Cost, p and q values for each link

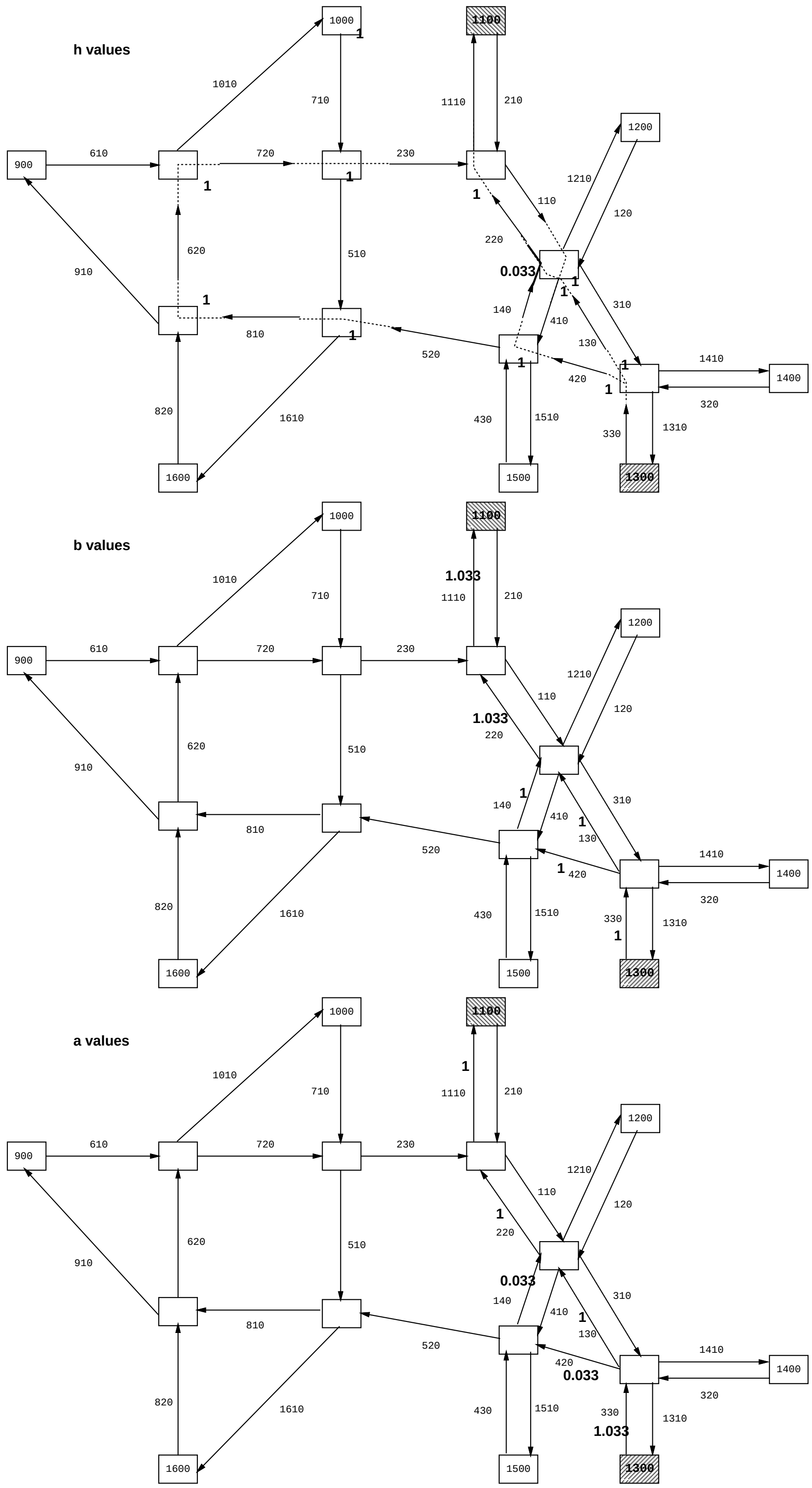


Figure 8: h, b, and a values for the Oxford Street example

6 SMART

Once the assignment matrix A and the turn probability vector P have been computed, one can solve the matrix equation $AX = P$ for the OD matrix X. We now detail the SMART algorithm [2] that we are using for this purpose.

6.1 What SMART Computes

When the matrix equation $AX = P$ has exactly one solution, SMART returns that solution, just as an ordinary method for solving uniquely determined linear systems of equations would.

When the matrix equation admits more than one solution, which is often the case since the number of OD pairs (unknowns) is typically greater than the number of turns (equations), SMART returns the solution with maximal entropy [9]. The rationale behind the maximum entropy solution is that it minimises the implicit information required in addition to that contained in A and P (which is insufficient) for obtaining a unique solution. Selecting any other solution would amount to constraining the available information more than we have to, and therefore to assume even more information that we do not have. We nevertheless leave the possibility of taking into account more information than A and P if one wishes to, by maximising entropy *relative to a user-provided estimate* of the OD matrix. So the algorithm takes as input A, P, and a $|\mathcal{P}|$-dimensional vector I, such that element I_p is an estimate of X_p. If no estimate is known at all, I_p may assign 0 to an OD pair $p = \langle o, d \rangle$ for which there is no path between o and d, and 1 otherwise. SMART can be seen as making the fewest unsupported changes to the estimate in order to satisfy the matrix equation. More formally, the entropy of X relative to I is defined as:

$$E(X, I) = -\sum_{p=1}^{|\mathcal{P}|} X_p \log(\frac{X_p}{I_p}) + X_p - I_p$$

and SMART returns the solution of $AX = P$ such that $E(X, I)$ is maximal. Note that if the estimate is already a solution, then its entropy is maximal.

When the matrix equation is inconsistent, meaning that the assumed driver route choice behavior model is inconsistent with the given turn probabilities, then SMART returns the OD matrix with maximal entropy $E(X, I)$ among those OD matrices closest to a solution. X is said to be closest to a solution when it maximises:

$$E(AX, P)$$

That is, the algorithm will select the OD matrix with maximal entropy, among the OD matrices that would require the fewest unsupported changes to P in order to be solutions.

6.2 The Algorithm

Figures 9 and 10 give the SMART algorithm. Note that it is possible to write shorter code implementing SMART, but that the version we give here exploits the fact that the A matrix is sparse. In particular, when we write:

$$\textbf{for } j \textbf{ from } 1 \textbf{ to } |\mathcal{P}| \textbf{ such that } A[i,j] \neq 0 \textbf{ do} \ldots \textbf{od}$$

we assume that only the non-zero elements $A_{i,j}$ will be retrieved.

The core of the algorithm is fairly simple (see Figure 10). It performs an iteration during which X (initially equal to I) is updated until convergence within a given ϵ is achieved. More precisely, if we note $\bar{X}$ the value of X at the previous iteration step, each element j of X, $1 \leq j \leq \mathcal{P}$, is updated as follows:

$$X_j \leftarrow \bar{X}_j \prod_{i=1}^{|\mathcal{T}|} \left(\frac{P_i}{\sum_{k=1}^{|\mathcal{P}|} A_{i,k} \bar{X}_k} \right)^{A_{i,j}}$$

with the convention that $X_j = 0$ whenever $\bar{X}_j = 0$, and that if the denominator $\sum_{k=1}^{|\mathcal{P}|} A_{i,k} \bar{X}_k$ equals 0, then the fraction equals 1. In the figure, we have decomposed this into three steps:

- computation of $G_i = \frac{P_i}{\sum_{k=1}^{|\mathcal{P}|} A_{i,k} \times \bar{X}_k}$ for each i, $1 \leq i \leq |\mathcal{T}|$
- computation of $N_j = \prod_{i=1}^{|\mathcal{T}|} (G_i)^{A_{i,j}}$
- update of $X_j = \bar{X}_j \times N_j$

The algorithm always converge, either to a solution if there is one, or else to an approximate solution. The boolean function $\mathsf{close}(X, \bar{X}, \epsilon)$ detects whether convergence is achieved, that is, whether X is close to its value at the previous iteration step $\bar{X}$ up to an ϵ. There are various ways one may want to implement this function (infinite norm, two norm ...). In our implementation, we chose to stop iterating when the two-norm of the change in X is smaller than ϵ, i.e.:

$$\mathsf{close}(X, \bar{X}, \epsilon) \equiv \sqrt{\sum_{j=1}^{|\mathcal{P}|} (X_j - \bar{X}_j)^2} \leq \epsilon$$

```
inputs:
      the |T| × |P| assignment matrix A
      the |T|-dimensional turn probability vector P
      the |P|-dimensional estimate I of the OD matrix
      the precision ϵ within which a solution is sought

output:
      the |P|-dimensional vector X representing the OD matrix

/* initialisation of X */
X ← copy(I)

/* X̄ contains the value of X at the previous iteration step */
/* initialisation of X̄ so that not close(X, X̄, ϵ)*/
for j from 1 to |P| do
      X̄[j] ← 1 + 2ϵ
od

/* rescaling of A and X, so that A's columns sum to 1 */
/* S[j] = Σ_{i=1}^{|T|} A[i, j] */
for j from 1 to |P| do
      S[j] ← 0
od
for i from 1 to |T| do
      for j from 1 to |P| such that A[i, j] ≠ 0 do
            S[j] ← S[j] + A[i, j]
      od
od
/* division of every column of A by the sum of its elements */
for i from 1 to |T| do
      for j from 1 to |P| such that A[i, j] ≠ 0 do
            A[i, j] ← A[i, j]/S[j]
            /* S[j] cannot be 0 unless A[i, j] is */
      od
od
/* multiplication of every element of X by the sum of the */
/* elements of the corresponding column of A */
for j from 1 to |P| do
      X[j] ← X[j] × S[j]
od
```

Figure 9: The SMART algorithm (initialisation and rescaling)

```
/* main loop */
while not close(X, X̄, ε) do
    X̄ ← copy(X)
    /* computation of G such that G[i] = P[i] / (Σ_k^{|P|} A[i,k] × X̄[k]) */
    for i from 1 to |T| do
        G[i] ← 0
        for k from 1 to |P| such that A[i,k] ≠ 0 do
            G[i] ← G[i] + A[i,k] × X̄[k]
        od
        if G[i] = 0 then
            G[i] ← 1
        else G[i] ← P[i]/G[i]
        fi
    od
    /* computation of N such that N[j] = Π_{i=1}^{|T|} (G[i])^{A[i,j]} */
    for j from 1 to |P| do
        N[j] ← 1
    od
    for i from 1 to |T| do
        for j from 1 to |P| such that A[i,j] ≠ 0 do
            N[j] ← N[j] × (G[i])^{A[i,j]}
        od
    od
    /* update of X */
    for j from 1 to |P| do
        if X̄[j] ≠ 0 then
            X[j] ← X̄[j] × N[j]
        /* otherwise X[j] already equals 0 */
        fi
    od
od

/* reverse scaling of X */
for j from 1 to |P| do
    if S[j] ≠ 0 then
        X[j] ← X[j]/S[j]
    /* otherwise X[j] equals 0 */
    fi
od
```

Figure 10: The SMART algorithm (main loop)

SMART only operates correctly when the A matrix has normalised columns, that is, when the elements of each column of A sum to 1. Therefore, prior to entering the main iteration, some rescaling of the original A matrix and of the X vector is necessary (see Figure 9). Let S_j be the sum of the elements in column j, that is for $1 \leq j \leq |\mathcal{P}|$:

$$S_j = \sum_{i=1}^{|\mathcal{T}|} A_{i,j}$$

We want to normalise A's columns by dividing the elements of each column j by S_j. Naturally, in order to preserve the original system of equation $P = AX$, we also need to multiply each X_j with S_j. Once this rescaled system has been solved, X has to be scaled back, by dividing each element X_j by S_j (see Figure 10).

The complexity of the initialisation phase of SMART and of an iteration step is proportional to the number of non-zero elements in A. We are not aware of any theoretical result concerning the rate of convergence of SMART. We will give experimental results in Section 8. Let us note, however, that we have observed that SMART converges much more quickly in the case where the matrix equation has a solution, and that for that particular case, even faster algorithms exist [6] (unfortunately, these latter do not produce any useful approximation in the inconsistent case). Note also that the acceleration of the convergence of algorithms like SMART is a current research topic, and preliminary results are encouraging [3].

7 Updating the Turn Probabilities

Once the OD matrix has been determined, updating the turn probabilities following an incident is very simple. We just need to change the features of the affected links, for instance the number of operational lanes on these links, re-compute the cost of these links, compute a new assignment matrix A' based on the new costs, and multiply this assignment matrix with the OD matrix X to get the turn probability vector P'. Recall that probabilities in P' are joint link probabilities. One may want to convert them back into conditional link probabilities Tr', which can be done as follows:

$$Tr'(l_1, l_2) = \frac{P'(\langle l_1, l_2 \rangle)}{\sum_{l \text{ s.t. } \langle l_1, l \rangle \in \mathcal{T}} P'(\langle l_1, l \rangle)}$$

The complexity of the whole process is roughly the same as that of computing the assignment matrix. Indeed, the complexity of multiplying the assignment matrix with the OD matrix is proportional to the number of non-zero elements in the assignment matrix.

Let us see the effect of an incident on the Oxford Street network. The original data (turn probabilities, flows, costs ...) are those in Tables 1 and 2. Assume that due to an incident on link 130, the number of operational lanes on that link is reduced to 1. This leads the cost of link 130 to increase from 10.36 to 13.20. Consequently, taking links 420 and 140 rather than link 130 when going from origin 1300 to destinations 1100 and 1200 should become more attractive than it used to (total cost of 13.79).

This is confirmed by the results given in Table 5, which contrasts the original turn probabilities with the updated ones. Turn probabilities that have changed as a result of the update are in bold face. As can be seen from the table, a greater percentage of cars are now turning onto link 420 from link 330, and the percentage of cars turning onto link 130 decreases accordingly (53.42% and 26,58%, vs 40% for both of them previously). As expected, among the number of cars on link 420, a greater percentage than before is now turning onto link 140. Note that the number of cars going from links 330 and 320 to link 520 via link 140 remains the same as before $((596 \times 91.68\% + 3134 \times 53.42\%) \times 60.45\% = (596 \times 90\% + 3134 \times 40\%) \times 75\%)$.

8 Experimental Results

In order to assess the performance of our approach in time and solution quality, we performed 3 types of experiments. These made use of the networks constituting the test suite for TRITRAM:

- Hong Kong: an arterial road in Kowloon,
- Sydney - Oxford Street: our previous example of the area around Oxford Street in Sydney,
- Sydney - George Street: the main north-south route through the central business district in Sydney,
- Albury: the central business district and route of the Hume highway in Albury,
- 3-Grid, an artificial 3 by 3 grid network (Figure 12 shows a 2-Grid). From each link, there is a probability p_s of going straight through, p_l of turning left, and p_r of turning right. We looked at the case where each of these are equal, i.e., $p_s = p_l = p_r = \frac{1}{3}$, and at the case where $p_s = 0.7$, $p_l = 0.2$, and $p_r = 0.1$.

The size of these networks in terms of number of nodes, links, turns and OD pairs is given in the first four columns of table 6. Note that for all of the networks,

l_1	l_2	$\mathrm{Tr}(l_1, l_2)$ (%)	$Tr'(l_1, l_2)$ (%)
120	310	25	25
120	410	75	75
220	1110	100	100
140	**220**	**65**	**73.44**
320	1310	5	5
420	**1510**	**20**	**16.12**
140	**1210**	**35**	**26.56**
320	**420**	**90**	**91.68**
520	1610	5	5
420	**520**	**75**	**60.45**
320	**130**	**5**	**3.32**
620	720	5	5
520	810	95	95
420	**140**	**5**	**23.43**
720	510	5	5
620	1010	95	95
820	910	10	10
720	230	95	95
820	620	90	90
110	310	50	50
210	110	100	100
110	410	50	50
310	1410	35	35
130	220	100	100
310	1310	65	65
230	1110	25	25
410	520	3	3
330	**420**	**40**	**53.42**
230	110	75	75
410	1510	97	97
510	810	25	25
330	**130**	**40**	**26.58**
430	520	20	20
510	1610	75	75
610	1010	5	5
710	230	5	5
330	1410	20	20
430	140	80	80
610	720	95	95
810	620	15	15
710	510	95	95
810	910	85	85

Table 5: Original and updated turn probabilities

town	nodes	links	turns	OD pairs	solution	time (s)
Hong Kong	8	14	18	25	exact	< 0.2
Sydney - Oxford St.	16	29	42	64	exact	0.3
3-Grid equip. turns	21	49	108	144	approximate	0.5
3-Grid	21	49	108	144	approximate	0.5
Sydney - George St.	74	148	211	676	approximate	3.1
Albury	74	184	291	1295	approximate	8.5

Table 6: SMART run-times

the number of OD pairs is always greater than the number of turns, and that therefore the matrix equation cannot be expected to have a unique solution.

The software for the experimentations includes a preliminary implementation of Dial's algorithm in Maple and a C implementation of SMART, which we ran on a Sparc Ultra 1 with 64Mb of memory. SMART was run with a precision $\epsilon = 10^{-5}$.

8.1 Data Consistency and SMART's Run Times

The first experiment aims at determining whether the available data and our driver route choice behavior model are consistent, and whether the run-time performance of SMART is acceptable for the needs of simulation. For each of the network examples, we ran SMART and recorded whether an exact solution was found (indicating that the data are consistent with Dial's route choice behavior model), and the run-time of the algorithm.

For two of the networks, Hong Kong and Oxford Street, SMART finds an exact solution to the equation $P = AX$ (see fifth column of Table 6), showing the compatibility of Dial's route choice behavior model and the available turn probability data. This implies that if we multiply the assignment matrix A with the OD matrix X returned by SMART, we get our original probability vector P back. In the other cases, we would only get and approximation of P. The purpose of the next experiment is to determine how good this approximation is.

Given that the OD matrix only has to be determined once at the beginning of the simulation (or more exactly once per period of time for which turn probability data are available), the run-times for SMART (last column of Table 6) are acceptable. Only networks of the size of Albury may really start to be challenging for simulation, depending on the type of application.

l_1	l_2	$Tr(l_1, l_2)$ (%)	$\tilde{Tr}(l_1, l_2)$ (%)
3-Grid equip. turns			
1110	2120	33.333	33.393
1120	1210	33.333	33.393
1240	1140	33.333	33.410
3020	3110	33.333	33.299
1310	2310	33.333	33.284
3230	2240	33.333	33.424
1440	1330	33.333	33.299
1120	1220	33.333	33.284
3-Grid			
1110	2120	20.000	15.041
1120	1210	10.000	13.595
1240	1140	70.000	74.472
3020	3110	10.000	9.991
1310	2310	70.000	67.196
3230	2240	20.000	14.509
1440	1330	10.000	9.991
1120	1220	70.000	67.196
Sydney - George St.			
2540	2030	70.000	67.770
1730	1310	50.000	44.535
3910	4910	50.000	56.344
3830	1120	75.000	75.441
3830	3930	25.000	24.558
220	5610	100.000	100.000
1020	1710	70.000	69.934
2330	2420	40.000	15.672
Albury			
150501	6000203	67.000	67.368
6000104	271101	100.000	100.000
150501	161202	33.000	32.631
192201	119220501	11.000	10.694
192201	132002	89.000	89.305
226604	50002	18.000	9.880
226702	226602	62.000	55.079
51702	6000803	44.000	71.165

Table 7: Quality of the approximation

8.2 Quality of the Approximation

When the available probability data are inconsistent with the driver route choice behavior model, the question of the quality of the approximation returned by SMART arises. We want to know to what extent P would have to be changed for the matrix equation $P = AX$ to have a solution. For the 3-grid, George Street and Albury networks, we therefore multiplied the the assignment matrix A given to SMART with OD matrix X returned by SMART, and compared the resulting approximate probability vector $\tilde{P}$ to the original P vector.

A subset of the results are given in Table 7, which shows, for a few selected turns $\langle l_1, l_2 \rangle$, the conditional link probability distributions Tr and $\tilde{Tr}$ corresponding to P and $\tilde{P}$ respectively[2]. As expected, the quality of the approximation really depends on the quality of the data. For the 3-Grid network with equiprobable turns, the data make reasonable sense given the structure of the network and Dial's route choice behavior model. Therefore the approximation is excellent and never exceeds 0.2 %. For the 3-Grid with non-equiprobable turns, it is obvious that the conditional link probabilities considered (0.7, 0.1, and 0.2) cannot reflect any sensible driver route choice behavior (just look at Figure 12 to be convinced). Therefore, the approximation is worse.

Considering non-artificial networks such as George Street and Albury, we find that there are some problems again with the data. A number of input links in the George Street network have a flow of zero. Other boundary effects lead a valid turn not to be part of any reasonable path linking any OD pair, hence creating an inconsistency. As for Albury, some of the turn probabilities really seem to have been made up. As a consequence, although the approximation is usually within a few percent, the difference between the two vectors is sometimes arbitrarily large. This rarely happens for the George Street network (see the last row), but is much less exceptional for Albury (see the three last rows).

8.3 Information content of turn probabilities

Even in the case where the matrix equation is solvable, we have seen that it will rarely have a unique solution. This means that the information contained in the turn probabilities is insufficient to determine a unique OD matrix. Our third experiment aims at pointing out when and to what extent this is the case. In this experiment, we chose or randomly generated an OD matrix and consistent link flows, determined the corresponding turn probabilities by multiplying the assignment matrix with that OD matrix, and looked at whether or not our approach could re-discover the original OD matrix from the turn probabilities and the assignment. Since the ultimate goal of our approach is to update the

[2]The conditional link probabilities are easier to apprehend than the joint ones.

	destination	
origin	1500	1300
900	20	10
1000	60	10

original OD matrix

	destination	
origin	1500	1300
900	24	6
1000	56	14

recovered OD matrix

Table 8: Original and recovered OD matrices

turn probabilities following a change in traffic conditions, we also compared the updated turn probabilities obtained from the original and reconstructed OD matrices, respectively, following a sample traffic incident.

8.3.1 When are turn probabilities (in)sufficient?

To see under what circumstances the turn probabilities contain less information than the OD matrix, consider for instance the Oxford Street network in Figure 11. Suppose a very simple OD matrix involving only 4 OD pairs: 20% of the cars go from origin 900 to destination 1500, 10% from origin 900 to destination 1300, 60% from origin 1000 to destination 1500, and the remaining 10% from origin 1000 to destination 1300 (see Table 8). We assume a total input flow of 1000 and the link flows given in the figure. The figure shows the resulting turn probabilities. The turn probabilities encode the fact that once the flows coming from the two origins have merged, 80% of cars are going to destination 1500 and 20% to destination 1300. But they have lost the information about the proportion of these coming from the two respective origins. In absence of other information, the solution maximising entropy assumes that the two respective origins make an identical contribution, i.e., 80% and 20% of their respective flow. This leads to a different OD matrix than the original one (again, see Table 8).

The same situation would occur with the grid network in Figure 12 taking, say, nodes 1 and 10 as origins and 2 and 13 as destinations. Naturally if we consider only one origin instead of two, the original OD matrix will be exactly recovered. It is not always easy to see immediately whether the turn probabilities contain enough information, without computing the rank of the system of equations. For instance, suppose that instead of taking nodes 1 and 10 as origins, we take 10 and 20. Assume that the OD matrix is the same as for the Oxford Street example, letting 2-Grid nodes 10, 20, 13, and 2 play the rôle of Oxford nodes 900, 1000, 1500 and 1300 respectively. Assume also the flows given in Figure 12. Contrary to what might be expected, the original OD matrix will be exactly recovered. Indeed, there are now two reasonable paths from one of the origins, namely 20, to each of the destinations, each of which going through opposite corners of the grid. These two paths being of equal cost, 50% of the traffic starting from origin 20 and going to a given destination will go through the upper right corner of

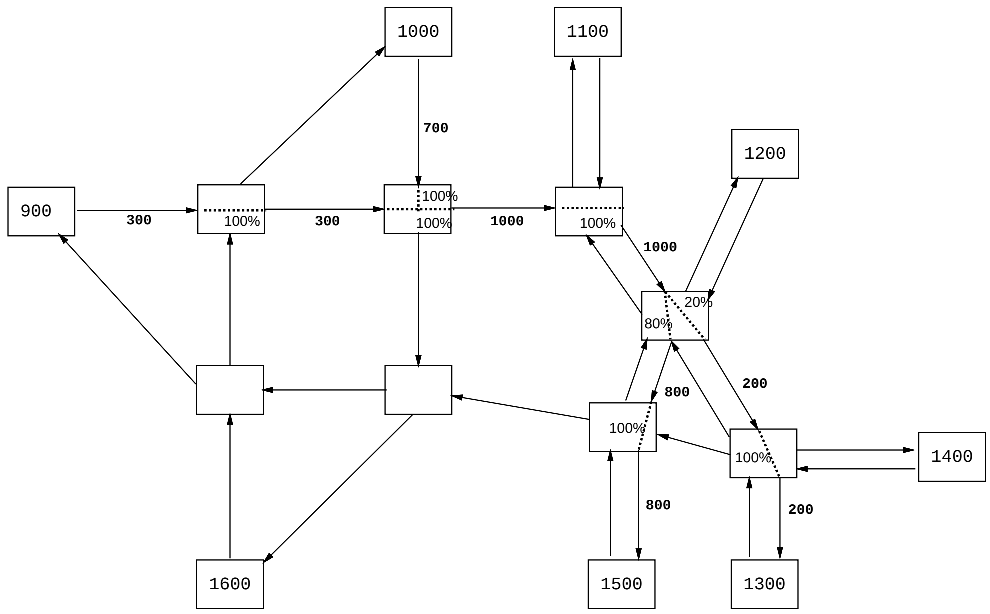


Figure 11: Turn probabilities computed from the original OD matrix

the grid, and the remaining half through the lower left corner. Therefore, the probability of going from origin 20 to destination 13 is twice the probability of taking turn t_1, that is $2 \times 0.7 \times 0.5 \times 1 \times 0.86 = 60\%$. Likewise, the probability of going from origin 20 to destination 2 is twice the probability of making turn t_2, that is $2 \times 0.7 \times 0.5 \times 1 \times 0.14 = 10\%$. Now, among the 20% of cars arriving at destination 2, we know that only 10% are coming from origin 20, therefore the remaining 10% are coming from origin 10. Likewise, since 60 of the 80% of cars arriving at destination 13 are coming from origin 20, therefore, 20% of the traffic is going from 10 to 2. That completes the reconstruction of the original OD matrix. Note that the SMART algorithm makes the same inferences.

8.3.2 Impact on OD matrix computation

In order to evaluate the effects of the lack of information, we randomly generated 50 OD matrices for the Oxford Street network, and recorded, for each trial, the distance between the recovered OD matrix and the original one.[3] We took the

[3]Note that, each time, we had to determine link-flows consistent with the OD matrix, since the link costs on which the assignment matrix is based depends on the flows. For that purpose, we first assigned equal flows to all links, determined an initial assignment matrix and turn probabilities, then used the probabilities to compute new flows, which were in turn determining a new assignment matrix and so on. We reiterated this process until convergence was achieved.

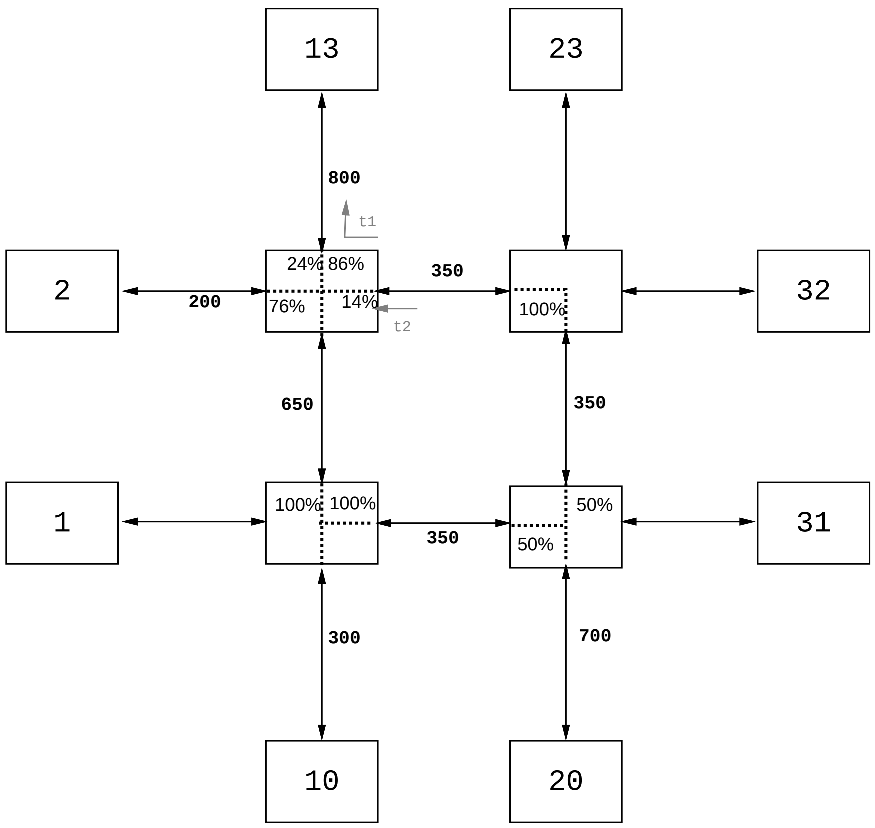


Figure 12: Turn probabilities computed from the original OD matrix for the 2-Grid network

distance between an original OD matrix X and its reconstruction $\tilde{X}$ to be:

$$\sum_{p \in \mathcal{P}} X_p |\log(\frac{X_p}{\tilde{X}_p})|$$

For a given OD pair p, the logarithm estimates the order of magnitude difference between the probabilities X_p and $\tilde{X}_p$ given by the two matrices, and this difference gets weighted by the original probability X_p of the OD pair. Compared to more straightfoward notions of distance (such as the average of the difference between the elements of the two matrices), this entropy-based distance has the advantage of giving a low importance to an OD pair with very low original probability, even if the difference of probabilities given by the two matrices is large. The reason is that not recovering this OD pair very accurately will not have a big impact on the simulation.

The result of this experiment is that, on average over the 50 trials, the distance between the original matrix and the recovered one is 0.13. Roughly, this means that on average, $\max(\frac{X_p}{\tilde{X}_p}, \frac{\tilde{X}_p}{X_p}) = 10^{0.13} \simeq 1.35$, and therefore, a probability mass of 35% is incorrectly recovered by the OD matrix computation process. This shows that even for a network like Oxford Street where the number of OD pairs is only 1.5 times larger than the number of turns, the information content of the turn probabilities is poor. A good prior estimation of the OD matrix can therefore be extremely useful (the present experiment was conducted without prior knowledge of the OD matrix).

8.3.3 Impact on turn probability updates

If the information content of the turn probabilities has a significant impact on the accuracy of the computed OD matrix, it turns out, on the other hand, that it has little effect on the accuracy of the updated probabilities.

To confirm this, we used the original and reconstructed OD matrices above to update the turn probabilities following a 50% increase of the cost of link 130 on the Oxford network, and compared the results obtained. That is, let A' be the assignment matrix following the increase of 130's cost, let as above X and $\tilde{X}$ be the original and reconstructed OD matrices, and let $P' = A'X$ and $\tilde{P}' = A'\tilde{X}$ be the updated turn probability vectors obtained from X and $\tilde{X}$ respectively, we recorded the distance:

$$\sum_{t \in \mathcal{T}} P'_t |\log(\frac{P'_t}{\tilde{P}'_t})|$$

Over the 50 trials, the average distance is only 5×10^{-5}, indicating roughly a shift of 0.01% of the probability mass.

9 Conclusion and Future Work

In this report, we have presented an approach to the update of existing traffic counts (turn probabilities, flows) following a change of traffic conditions. This approach attempts to construct a plausible OD matrix reflecting the available data and Dial's model of driver route choice behavior. The probabilities are then updated by making them consistent with this OD matrix and an assignment reflecting the new traffic conditions.

In addition to answering our initial problem, this method addresses several other interesting issues as a by-product:

- We can test the existing traffic counts against a reasonable driver route choice behavior model. It suffices to check whether the matrix equation is solvable.
- We can extract an OD-matrix from possibly incomplete traffic counts. Indeed, this constitutes the first step of our approach. Although this report has not exploited the fact that the matrix equation could well have less equations than the number of turns, the approach allows for turn counts to be missing.
- When the traffic counts are highly incomplete or when their information content is weak, it might be best to use them to improve an OD matrix estimate than to compute the OD natrix from scratch. This also can be achieved within our framework.
- The approach can be used to infer complete turn probabilities from the values of the flows and partial turn counts. Indeed, this amounts to computing an OD matrix consistent with the available data, and to inferring the turn probabilities from this OD matrix and the assignment.

The purpose of the current Maple implementation having been to test the feasibility of the approach, additional work is needed to build a efficient fully operational version and link it to the TRITRAM simulator. Following this, further experimentations will be needed in order to validate the proposed method on real incident data.

Linking the implementation to TRITRAM will also enable us to use TRITRAM link travel times in place of the link cost model adopted in the present report, leading to more accurate costs.

We believe that our method could easily be extended to simulate equilibrium assignment. Once the OD matrix has been determined, nothing forces us to update the probabilities in one step. We can update them iteratively: we can start by determining a new turn probability vector as previously, then run the simulation with the new probabilities for a little while, use the resulting TRITRAM travel-times to compute a new assignment[4], update the probability vector again in accordance to this new assignment, and so on. While the iteration is not guaranteed to converge towards a unique turn probability vector, this may make the simulation smoother and more realistic in many cases.

[4]Or, when the approach is used in conjunction with a flow-dependent link cost model as in our current implementation, the turn probabilities determine new flows which in turn determine new costs

References

[1] G. J. Both and C. Bayley. Evaluation procedures for rural road and structure projects. In *Proc. 8th ARRB Conference*, pages 6–25, 1976.

[2] C. Byrne. Iterative image reconstruction algorithms based on cross entropy minimization. *IEEE Transactions on Image Processing*, 2(1):96–103, january 1993.

[3] C. Byrne. Convergent block-iterative algorithms for image reconstruction from inconsistent data. *IEEE Transactions on Image Processing*, 6(9):1296–1304, september 1997.

[4] R. B. Dial. A probabilistic multipath traffic assignment model which obviates path enumeration. *Transportation Research*, 5:83–111, 1971.

[5] E.W. Dijkstra. A note on two problems in connection with graphs. *Numerical Math*, pages 269–271, 1959.

[6] R. Gordon, R. Bender, and G. Herman. Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography. *Journal of Theoretical Biology*, 29:471–481, 1970.

[7] S. Nguyen. Estimating origin-destination matrices from observed flows. In M. Florian, editor, *Transportation Planning Models*, pages 363–380. Elsevier, Amsterdam, 1984.

[8] V. N. Nguyen, P. Kilby, and P. Lamb. Validation of the TRITRAM traffic network simulation model. In *Proc. 3rd International Conference of ITS Australia (ITSA-97)*, Brisbane, Australia, 1997.

[9] C. E. Shannon. A mathematical theory of communication. *Bell System Technology Journal*, 27:379–423, 623–656, 1948.

[10] J. H. Van Zuylen and L. G. Willumsen. The most likely trip matrix estimated from traffic counts. *Transportation Research*, 14B:281–293, 1980.

[11] J. G. Wardrop. Some theoretical aspects of road traffic research. In *Proc. Inst. Civil Eng.*, pages 325–378, 1952. Part II, 1.