THE RECURSIVE LOGIT MODEL
TUTORIAL
CIRRELT SEMINAR
August 2017

MAELLE ZIMMERMANN*, TIEN MAI*, EMMA FREJINGER*

CIRRELT and Université de Montréal, Canada
Ecole Polytechnique de Montréal, Canada
TUTORIAL GOALS

- The recursive logit model is a random utility model for the choice of path in a network with no restriction on the choice set. It is based on dynamic discrete choice theory.
- This tutorial is focused on theory and practice and aims to
  - describe the problem, the model and its advantages
  - familiarize the reader with the open-source Matlab code
We assume the reader is familiar with discrete choice theory and refer to the textbook of Ben-Akiva and Lerman for insights on this topic.

OUTLINE

▶ Introduction
▶ Theory
  ▶ Route choice modeling approaches
  ▶ Recursive logit model formulation
  ▶ Maximum likelihood estimation
▶ Practice
  ▶ Matlab
  ▶ Code structure
  ▶ Data files
  ▶ A first example
1 Introduction: the problem

2 Theory: the models

3 Practice: making it work
ROUTE CHOICE MODELING

The problem

▶ Given an origin and destination in a transport network, which route does a traveler choose?
▶ Travelers do not always choose the shortest path in terms of distance
▶ Other attributes affect the choice through a generalized cost function
MODELING FRAMEWORK

Discrete choice framework

- Analyst observes path choices but has imperfect knowledge of travelers’ generalized cost and perception of network
- Parameters to be estimated on such data describe individuals’ preferences for attributes
- The estimated models define choice probability distributions over alternative paths
OBJECTIVES

A model that can be

- **Consistently estimated** in reasonable time using path choice
data collected in real large-scale networks
- used for **accurately predicting** path choices in short
computational time (e.g. in a traffic simulation context)
WHY IT IS DIFFICULT

<table>
<thead>
<tr>
<th>The choice set problem</th>
<th>The correlation problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ We don’t know <strong>what alternatives</strong> individuals consider and there are infinitely many paths connecting each OD pair</td>
<td>▶ Many paths <strong>overlap</strong> in a real network and overlapping paths probably share unobserved attributes</td>
</tr>
</tbody>
</table>
1. Introduction: the problem

2. Theory: the models

3. Practice: making it work
ROUTE CHOICE MODELS

Route choice models can be categorized according to the way they deal with choice sets.

- Consideration sets
  - Generated choice sets treated as true ones
  - Parameter estimates differ depending on choice set
  - Issues with prediction

- Universal choice sets
  - Sampling of alternatives
  - Consistent estimates
  - Issues with prediction

- Recursive logit
  - Consistent estimates
  - Straightforward for prediction
"CONSIDERATION SETS" APPROACH

- Choice of path modeled as *selection from a discrete set of routes*
- Since the set of feasible routes between $O$ and $D$ cannot be enumerated, the modeler generates a subset of path alternatives.
- The generated choice set is treated as the true one.
- **Problem:** Parameter estimates may significantly vary depending on the choice set definition!
ANOTHER APPROACH

Recursive logit model

- Choice of path is formulated as a sequence of link choices
- A specific case of dynamic discrete choice
DYNAMIC DISCRETE CHOICE

- Horizon $T$, steps $t = 1, ..., T$
- Individuals make **sequential decisions** over horizon
- A **state** $s_t$ describes all the information relevant for the individual at step $t$
- An **action** $a_t$ is the decision taken at step $t$, affecting the value of future state $s_{t+1}$
- Instantaneous **utility** $u(a_t|s_s)$ of choosing action $a_t$
- A **Markov transition density function** $F(s_{t+1}|a_t, s_t)$ describes the evolution of future states
DYNAMIC DISCRETE CHOICE

- **Unobserved** state variables $\varepsilon_t$, **observed** state variables $x_t$
- $\varepsilon_t$ is an i.i.d. extreme value type I random variable
- Utility separable in $u(a_t|s_t) = v(a_t|x_t) + \varepsilon_t$
- $v(a_t|x_t)$ parametrized by $\beta$ to be **estimated**
- Markov transition function becomes $F(\varepsilon_{t+1})F(x_{t+1}|a_t,x_t)$
- Individual chooses action $a_t$ which maximizes instantaneous and future utility
- **Expected maximum utility** at state $x_t$ given by integrated Bellman’s equation

$$V(x_t) = E_{\varepsilon} \left[ \max_{a_t} \left( v(a_t|x_t) + \varepsilon_t + \int V(x_{t+1})dF(x_{t+1}|a_t,x_t) \right) \right]$$
RECURSIVE LOGIT MODEL

A dynamic discrete choice model for the choice of path

- The network is represented by a graph $G = (A, V)$
- A state $k \in A$ is a link in the network
- An action $a \in A(k)$ is an outgoing link at the sink node of $k$
- Infinite horizon but absorbing link $d$ with no successor corresponding to destination.
- A path is a sequence of states $k_0, ..., k_I$ with $k_{i+1} \in A(k_i)$ $\forall i$ and $k_I = d$
ATTRIBUTES

- We denote $x(a|k)$ the attributes of the link pair $(k,a)$
- Parameters $\beta$ describe individual preferences regarding attributes
- $u(a|k)$ is the random utility of link $a$ given current link $k$
- $u(a|k) = v(a|k) + \mu \varepsilon(a), \varepsilon(a)$ i.i.d EV type I
- $v(a|k) = \beta^T x(a|k)$
- Attributes must be link-additive and deterministic
LINK CHOICE SITUATION

- Traveler chooses next link $a$ given current state $k$
- Next state $k_{t+1}$ is **given with certainty** by the action $a_t$ since $k_{t+1} = a_t$
- Traveler chooses action $a \in A(k)$ that maximizes sum of $u(a|k)$ and expected maximum utility to destination $V^d(a)$, denoted the **Value function**
- Link choice probabilities

- Value function to destination given by Bellman’s equation

\[ V^d(k) = E_\varepsilon \left[ \max_{a \in A(k)} \left\{ v(a|k) + V^d(a) + \mu \varepsilon(a) \right\} \right] \]  

(1)

- Link choice probability given by logit model

\[ P^d(a|k) = \frac{e^{\frac{1}{\mu} v(a|k) + V^d(a)}}{\sum_{a' \in A(k)} e^{\frac{1}{\mu} v(a'|k) + V^d(a')}}. \]  

(2)
The exponential of the Value function in (1) can be rewritten

\[ e^{\frac{1}{\mu} V(k)} = \begin{cases} \sum_{a \in A} \delta(a|k) e^{\frac{1}{\mu} (v(a|k) + V(a))} \quad &\text{if } k \in A, \\ 1 \quad &\text{if } k = d. \end{cases} \]

The Value function is obtained by solving a system of linear equations

\[ z = Mz + b \iff z = (I - M)^{-1} b \]

where

\[ z_k = e^{\frac{1}{\mu} V(k)} \]

\[ b_k = 0 \quad \forall k \in A, \quad b_d = 1 \]

\[ M_{ka} = \delta(a|k) e^{\frac{1}{\mu} v(a|k)} \]
Path choice probabilities

Path $\sigma = k_0, ..., k_I$ where $k_I = d$ with choice probability

$$P(\sigma) = \prod_{i=0}^{I-1} P^d(k_{i+1} | k_i)$$

$$= \frac{e^{\frac{1}{\mu} \sum_{i=0}^{l-1} v(k_{i+1} | k_i)}}{e^{\frac{1}{\mu} V^d(k_0)}}$$

$$= \frac{e^{\frac{1}{\mu} v(\sigma)}}{\sum_{\sigma' \in \mathcal{U}} e^{\frac{1}{\mu} v(\sigma')}}.$$ 

The RL model is equivalent to a static multinomial logit model with universal choice set $\mathcal{U}$. 
WHY IS THE RL BETTER?

Advantages over path-based models

- Avoids generating choice sets of paths both for estimation and prediction
- Parameter estimates are consistent
- Efficient for prediction
MAXIMUM LIKELIHOOD ESTIMATION

- Data of observed path choices $\sigma_n$, $n = 1, \ldots, N$.
- Maximum likelihood estimation problem

$$\max_{\beta} \sum_{n=1}^{N} \ln P(\sigma_n; \beta)$$

- Estimation requires to combine inner and outer algorithm, e.g. Nested Fixed Point (NFXP) algorithm
  - Outer algorithm: solves the non-linear optimization problem
  - Inner algorithm: solves the Value functions
MAXIMUM LIKELIHOOD ESTIMATION

\[ \hat{\beta} \]

Outer loop: 2nd order optim. algo.

Inner loop: Bellman equation

Converged?

get \( V(\beta) \)

\[ \beta_0 \]

no

yes

update \( \beta \)
PREDICTION

- Depending on applications, it may be useful to
  - Sample a path from the estimated distribution (e.g. applications with scenarios)
  - Predict expected link flows assuming a fixed demand (e.g. traffic assignment applications)

Advantages of RL model

- Allows path sampling without choice set generation
- Paths sampled according to the true estimated probabilities
- Possibility to fastly compute expected link flows without repeated path sampling
PREDICTION

Path sampling
- Sequentially sample arcs $k_0, k_1, \ldots$ until reaching arc $d$
  according to estimated link choice probabilities $P^d(a|k)$
  $\forall k, a \in A$ in (2)

Expected link flows
- $P^d$: link choice probabilities $\forall k, a \in A$
- $G^d$: demand originating at $a \in A$ and ending at $d$
- $F^d$: expected flow towards $d$ on $a \in A$ obtained by solving

$$F^d(a) = G^d(a) \sum_{k \in A} P^d(a|k)F(k)$$

(3)
LINK SIZE ATTRIBUTE

- Similar to Path Size attribute for path-based models
- Heuristically **corrects utility** of overlapping paths

Computing the LS attribute

- Choose utility with parameters $\tilde{\beta}$
- For each OD pair, compute expected link flow $F^{OD}$ with (3) where $G$ is zero-valued except for $G(O) = 1$.
- Link size attribute is expected link flow

$$LS^{OD} = F^{OD}$$

- Note: the LS attribute is **origin-destination specific!**
1 Introduction: the problem

2 Theory: the models

3 Practice: making it work
MATLAB

- The implementation of the RL model is in MATLAB
THE CODE

- The Recursive logit code is available on GitHub here: https://github.com/maitien86/RL-Tutorial
- You can clone the following repository
  $ git clone git://github.com/maitien86/RL-Tutorial
This tutorial is aimed at users who want to use the code to estimate a path choice model with their own data.

We will go through the type of input data needed and the main functions of the code.

We will show how the code works on an illustrative dataset.
QUICK OVERVIEW

- You will need to handle the following files
  - `loadData.m`
    Loads the network data and observations given in the Input folder.
  - `initializeOptStruct.m`
    Tunes estimation algorithm and model parameters.
  - `RLoptimizer.m`
    Begins the maximum likelihood estimation algorithm and returns estimates.
ILLUSTRATIVE DATASET

Figure: Example network labeled with link IDs
DATA FILES

- The code requires the following data files to be placed in the Input folder:
  - LinkAttributes.txt
    A file containing attributes values for all links
  - Incidence.txt
    The matrix representation of the graph $G$
  - Observations.txt
    Link-by-link descriptions of observed itineraries
LINK ATTRIBUTES

- This file describes the attributes of each network link. It consists of several columns containing each an attribute value. The first three columns should indicate:
  - The link ID,
  - The ID of the front node,
  - The ID of the end node.

- In practice for real networks such a file can often be obtained from GIS data.

Figure: Link attributes file for example network
The LinkAttributes matrix can be read from the link attributes file

Reading the link attributes file

```matlab
file_linkAttributes='./Input/LinkAttributes.txt';
linkAttributes = csvread(file_linkAttributes,1,0);
```
This file describes the incidence matrix of the graph $G$. In practice it can be directly generated from the LinkAttributes matrix.

**Generate incidence matrix**

```matlab
nLinks = length(LinkAttributes(:,1));
A = LinkAttributes(:,3);
B = LinkAttributes(:,2);
Incidence = sparse(nLinks,nLinks);
for i = 1:nLinks
    U = find(B == A(i));
    Incidence(i,U) = 1;
end
```
INCIDENCE MATRIX

- The Incidence matrix should also include **dummy links** for each observed destination.
- In the illustrative example we have 29 network links and we consider a single destination corresponding to node 17.
- An absorbing link (here labeled 30) should be added to the set of network links.
- An added column should be added to the Incidence matrix, of final size $29 \times 30$. 
The Incidence matrix can be saved as a text file so it can be easily loaded for future use.

Save and load the incidence matrix

```matlab
[i,j,val] = find(Incidence);
data_dump = [i,j,val];
save('IncidenceMatrix.txt','data_dump','-ascii');
```

```matlab
file_incidence='./Input/IncidenceMatrix.txt';
Incidence = spconvert(load(file_incidence));
```
This file describes observed trajectories in terms of sequences of link IDs.

The destination link should be repeated at the beginning of the sequence.

Sample of observations in example network

<table>
<thead>
<tr>
<th>Obs.</th>
<th>dest.</th>
<th>orig.</th>
<th>links</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>1</td>
<td>3 14 6 26 30</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>1</td>
<td>3 5 8 11 29 30</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1</td>
<td>3 5 8 28 30</td>
</tr>
<tr>
<td>4</td>
<td>30</td>
<td>1</td>
<td>3 5 8 28 30</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>1</td>
<td>12 21 25 30</td>
</tr>
</tbody>
</table>
OBSERVATIONS

- In practice data processing steps may be required to obtain observed data in the desired format.
- Similarly to the Incidence matrix, observations should be saved as a text file that can be loaded as a sparse matrix Obs.

**Save and load observation matrix**

```matlab
[i,j,val] = find(Obs);
data_dump = [i,j,val];
save('Observations.txt','data_dump','-ascii');
```

```matlab
file_incidence='./Observations.txt';
Obs = spconvert(load(file_observations));
```
SPECIFYING ATTRIBUTES

Attributes are specified in the `loadData.m` file. Since the RL model requires link pairs attributes, this require several steps.

1. extract from the `LinkAttributes` matrix the attribute columns to be included in the model specification,
2. transform link attributes vectors into link pair attributes matrices,
3. store each attribute matrix into the `Atts ObjArray` variable.
EXAMPLE SPECIFICATION

- In the illustrative example, we specify 3 link pair attributes
  - Link length
  - Presence of traffic signal
  - Link constant

- Attributes are defined for link pairs \((k, a)\) and may be independent of state \(k\) (e.g. link length of \(a\)) or dependent on both states (e.g. turn angle between links \(k\) and \(a\))
**SETTING PARAMETERS**

▶ Parameters are set in the `initializeOptStruct.m` file

▶ The first set of parameters to tune is related to the estimation algorithm.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op.OptimMethod</td>
<td>Whether to use a line search or trust region method</td>
</tr>
<tr>
<td>Op.HessianApprox</td>
<td>Whether to use a BFGS or BHHH Hessian approximation</td>
</tr>
<tr>
<td>Op.maxIter</td>
<td>The maximum number of iterations of the algorithm</td>
</tr>
</tbody>
</table>

▶ The second set consists of model parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op.n</td>
<td>The number of attributes in the utility specification</td>
</tr>
<tr>
<td>Op.LinkSize</td>
<td>A boolean to include or not a Link Size attribute</td>
</tr>
</tbody>
</table>
RUNNING THE ESTIMATION ALGORITHM

- The main file RLoptimizer.m starts the model estimation procedure
- The files loadData.m and initializeOptStruct.m are called within this file
- Details of each iteration are reported
- The value of estimated parameters and standard deviation are displayed at the end and can optionally be saved in the Results folder
The algorithm stops, due to RELATIVE GRADIENT

The attributes are

[Iteration]: 15

LL = 2.381999

x =

-2.358565e+00
-3.327645e-01
2.640459e-02

norm of step = 0.000150
radius = 0.000601
Norm of grad = 0.0000002
Norm of relative gradient = 0.000000
Number of function evaluation = 20.000000

Number of function evaluation 20

Estimated time 5.739890e-01

Figure: Output of model estimation for illustrative example