Université de Montréal

On Inverse Reinforcement Learning and Dynamic Discrete Choice for Predicting Path Choices

par

Drew Kristensen

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

November 30, 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

On Inverse Reinforcement Learning and Dynamic Discrete Choice for Predicting Path Choices

présenté par

Drew Kristensen

a été évalué par un jury composé des personnes suivantes :

Simon Lacoste-Julien
(président-rapporteur)

Emma Frejinger
(directeur de recherche)

Fabian Bastin
(membre du jury)

Résumé

La modélisation du choix d'itinéraire est un sujet de recherche bien étudié avec des implications, par exemple, pour la planification urbaine et l'analyse des flux d'équilibre du trafic. En raison de l'ampleur des effets que ces problèmes peuvent avoir sur les communautés, il n'est pas surprenant que plusieurs domaines de recherche aient tenté de résoudre le même problème. Les défis viennent cependant de la taille des réseaux eux-mêmes, car les grandes villes peuvent avoir des dizaines de milliers de segments de routes reliés par des dizaines de milliers d'intersections. Ainsi, les approches discutées dans cette thèse se concentreront sur la comparaison des performances entre des modèles de deux domaines différents, l'économétrie et l'apprentissage par renforcement inverse (IRL).

Tout d'abord, nous fournissons des informations sur le sujet pour que des chercheurs d'un domaine puissent se familiariser avec l'autre domaine. Dans un deuxième temps, nous décrivons les algorithmes utilisés avec une notation commune, ce qui facilite la compréhension entre les domaines. Enfin, nous comparons les performances des modèles sur des ensembles de données du monde réel, à savoir un ensemble de données couvrant des choix d'itinéraire de cyclistes collectés dans un réseau avec 42 000 liens.

Nous rapportons nos résultats pour les deux modèles de l'économétrie que nous discutons, mais nous n'avons pas pu générer les mêmes résultats pour les deux modèles IRL. Cela était principalement dû aux instabilités numériques que nous avons rencontrées avec le code que nous avions modifié pour fonctionner avec nos données. Nous proposons une discussion de ces difficultés parallèlement à la communication de nos résultats.

Mots-clés: Modélisation de choix d'itinéraire, Prévision des flux de trafic, Modèles de choix discret dynamique, Apprentissage par renforcement inverse

Abstract

Route choice modeling is a well-studied topic of research with implications, for example, for city planning and traffic equilibrium flow analysis. Due to the scale of effects these problems can have on communities, it is no surprise that diverse fields have attempted solutions to the same problem. The challenges, however, come with the size of networks themselves, as large cities may have tens of thousands of road segments connected by tens of thousands of intersections. Thus, the approaches discussed in this thesis will be focusing on the performance comparison between models from two different fields, econometrics and inverse reinforcement learning (IRL).

First, we provide background on the topic to introduce researchers from one field to become acquainted with the other. Secondly, we describe the algorithms used with a common notation to facilitate this building of understanding between the fields. Lastly, we aim to compare the performance of the models on real-world datasets, namely covering bike route choices collected in a network of 42,000 links.

We report our results for the two models from econometrics that we discuss, but were unable to generate the same results for the two IRL models. This was primarily due to numerical instabilities we encountered with the code we had modified to work with our data. We provide a discussion of these difficulties alongside the reporting of our results.

Keywords: Route choice modeling, Traffic flow prediction, Dynamic discrete choice models, Inverse reinforcement learning

Contents

List of tables

List of figures

List of acronyms and abbreviations

DDC	Dynamic Discrete Choice
DP	Dynamic Programming
DSP	Deterministic Shortest Path
GMCE	Generalized Maximum Causal Entropy
IIA	Independence of Irrelevant Alternative
IOC	Inverse Optimal Control
IRL	Inverse Reinforcement Learning
ISP	Inverse Shortest Path
LL	Log Likelihood
MDP	Markov Decision Process
ME	Maximum Entropy
ML	Machine Learning
MNL	Multinomial Logit
NFXP	Nested Fixed Point
NRL	Nested Recursive Logit
PM	Path Matching
POMDP	Partially Observable MDP
RCM	Route Choice Model
RUM	Random Utility Maximization
SSP	Stochastic Shortest Path
VI	Value Iteration

Remerciements

There are so many people who have helped through this thesis, either by giving me support and motivation or by providing assistance, or both. Of course, my deepest gratitude for my thesis advisor, Dr. Emma Frejinger. She has not only helped me through the ups and downs of this process, but the advice provided by her has been invaluable. Without her guidance, I am sure this thesis would not have been completed. I also must thank Dr. Tien Mai, who provided me resources to assist with the implementation of the models used. To Céline Bégin, thank you for being patient and understanding, and being my ally in navigating a francophone system and always looking out for my best interests.

To my family, thank you for cheering me on over the past few years and believing that I would finish. All the various kinds of support you gave me are the reason I was able to see this through and finally complete this. To my mother and father, thank you especially for everything, from checking in with me each week to helping me move across the country during a pandemic to the little reminders of your care. To my Montreal family, thank you for always being the friendly bunch you are, and keeping my chin up when things were rough. You made my time in Montreal incredibly special and I could never have stayed so strong without all your encouragement and love. To Shawn, thank you for being such an amazing friend and being a beacon of light throughout my time there. To my friends, thank you for the confidence you placed in me and the constant reassurance that I would be able to complete my program. All of you helped me in one way or another, be it through taking my mind off things when everything was stressful, making sure I was still eating, or just being a vent for my frustrations.

To all these people, I am so deeply thankful for, and this thesis is not my accomplishment alone, but only possible due to all of them. Thank you.

Chapter 1

Introduction

While the modern car has improved connectivity between both individuals and communities, there is no doubt that the associated traffic that comes alongside a high volume of cars is undesirable. While the obvious negative for any driver who has been stuck in rush hour traffic would be the time delays, there are many other issues that arise with vehicular traffic, namely noise, environmental pollution, and traffic accidents. In order to minimize these negative factors, cities employ urban planners and network administrators to design and plan effective transport networks. Their jobs are to maximize the total utility for all occupants on the road, whereas each individual tends to make decisions based on what they believe will maximize their individual utility. This discrepancy of goals between individuals and their community exposes an important problem and has been a well-researched area, specifically part of what is known as route choice modeling.

Route choice modeling seeks to understand the choices that travelers in a network make. This understanding is gained through the ability to both predict the route a traveler will take as well as explaining why the traveler made that choice. While being able to explain past actions may have some benefit, the primary goal of route choice models (RCMs) is to tackle the problem of what is called counterfactual prediction or policy forecasting in econometrics. That is, the prediction of how traffic operates under different, novel scenarios. These predictions are central to many different transport problems, such as traffic equilibrium analysis. The interpretation of parameter values (e.g., value of time) along with the predictions allow experts such as city planners to better design their road system to best serve the community under a wide variety of possible settings.

The difficulties related to RCM boil down to three requirements stated by Zimmermann and Frejinger (2020); firstly, the route choice model must be scalable to large networks that reflect the scale of modern day transportation systems. Secondly, the models must be generalizable to different scenarios (the counterfactual prediction we mentioned a moment ago), thus giving benefit to the model beyond the inherently limited number of scenarios

the data was gathered under. Lastly, these models must be interpretable, so the effects of different features within the network can be understood. For example, Ziebart et al. (2008) found the cost to a driver when taking a left turn across traffic to be approximately 6 seconds of travel time.

With such an important problem, it is no surprise that diverse fields have made simultaneous progress in the attempt to find novel solutions. Specifically, the field of econometrics has utilized a litany of discrete choice models, culminating in approaches such as the recursive logit model and nested recursive logit models. Another field, machine learning (ML), has also attempted solutions within the subfield of inverse reinforcement learning (IRL), with algorithms such as maximum entropy IRL and its generalized counterpart, generalized maximum causal entropy. However, there has been little cross-over between the two fields, in spite of the similarity of their approaches and applicability to each other's fields. Creating this explicit link is the primary goal of this thesis. By linking the two fields together, through rewriting algorithms with common notation as well as comparing the performances of the different algorithms, we hope to open up a bridge between the literatures and provide an introduction to researchers from one area about the other.

The remainder of the thesis is structured as follows: Chapter 2 provides background information on both the econometrics approach as well as the background for understanding the inverse reinforcement learning approach for someone from the opposite field. This section should be sufficient in bringing the reader quickly up to speed on what they should be aware of before reading the algorithms and models specific to the route choice problem. Chapter 3 discusses the methodologies of the two fields. For each field, we describe how the algorithms work, provide pseudo-code for the algorithms, discuss why we chose to focus on the algorithms in this thesis, and lastly, discuss the metrics used and a justification of their usage. Chapter 4 presents numerical results for the four models used and a discussion on the impact of the results.

Chapter 2

Background on Dynamic Discrete Choice and Inverse Reinforcement Learning

Before describing the algorithms and models that this thesis will focus on, it is first important to understand their background. First, we introduce Discrete Choice Models, the framework that our algorithms utilize. To explain their workings, we discuss random utility maximization and the property of Independence of Irrelevant Alternatives. Next, we discuss the area of shortest path problems, and we introduce the reader to the concept of Markov Decision Processes (MDP) as a useful setting for sequential decision problems. Thirdly, we discuss the area of dynamic discrete choice models, describing their parameter estimation as well as introducing the Multinomial Logit Model which we build on in Chapter 3. Lastly, we introduce the reader to Inverse Reinforcement Learning, with much the same approach as with the dynamic discrete choice models.

2.1. Discrete Choice Models

Studied within the field of econometrics, Random Utility Maximization (RUM) models are designed to describe how actors in an environment make choices given a set of information and available actions. The framework of any RUM model is that individual n makes a choice p from a choice set C_n, based on the utility of a choice, $u_n(p)$. A choice set C_n is the set of all available choices for individual n. The utilities are indexed by individual as the utility from one traveler to another may be different, as they may value different aspects of any action with different weights. The utility can be viewed as the assessment of how good a choice is for an individual. Numerically, we express utility as the variable $u_n(p)$, where $u_n(p) = v_n(p) + \mu\varepsilon_n(p)$, with $v_n(p)$ representing the deterministic utility of choice p, and $\mu\varepsilon_n(p)$ representing the random utility of choice p. The deterministic utility $v_n(p)$ encompasses the measured attributes of the choice or attributes specific to the individual, whereas the random utility $\mu\varepsilon_n(p)$ covers the uncertainty in the observed data, where the μ

is a scale parameter of the random variable. Commonly, the deterministic utility is linear-in-parameters, and is written as $v_n(p) = \theta \mathbf{x}_n(p)$, with θ being a parameter vector with coefficients to be estimated, and $\mathbf{x}_n(p)$ being a vector of attributes containing the information on the choice p for individual n. We assume that individuals seek to maximize this utility with respect to their choices, a concept known as "utility maximization". Under this assumption, we can formulate the probability of an individual n choosing path p from C_n as the following:

$$\begin{aligned} P(p|C_n) &= P(u_n(p) > u_n(p') \ \forall \ p' \in C_n) \\ &= P(u_n(p) = \max_{p' \in C_n} u_n(p')). \end{aligned}$$

While there are a wealth of different types of models that depend on the assumptions made on distribution of the random error terms, by assuming the error terms $\varepsilon_n(p)$ are independent and identically distributed (i.i.d.) extreme value type I, we get the Multinomial Logit (MNL) formulation (McFadden, 1977). This is one of the more simple assumptions and provides the basis for the later models discussed, though as can be seen in Train (2009), there exist many other assumptions and models that could be used in a MNL's stead. With the MNL, we can reformulate our previous probability of taking choice p to be

$$P(p|C_n) = \frac{e^{v_n(p)}}{\sum_{p' \in C_n} e^{v_n(p')}}.$$

In the study of RUMs, it is critical to consider alternatives when analysing which choice to make. For example, the MNL model exhibits the Independence of Irrelevant Alternatives (IIA) property. It is useful because it ensures that relative likelihood between two options is unaffected by the addition of an alternative. For example, if the majority of voters preferred candidate A to candidate B, then when a novel candidate C is introduced, B should not now be preferred to A. While this property seems logical, there are certain issues with the IIA property. The textbook case of problems arising with this would be an individual choosing between a car and a blue bus McFadden (1977). If the individual has no preference, having a (1:1) ratio of choosing one against the other, then upon allowing their choice to include a red bus, the IIA property imposes that the resulting odds for the individual must keep the (1:1) odds for the car and blue bus. However, to many individuals, there may be no difference in choice between a red and blue bus, and thus the odds of choosing car, blue bus, red bus may take the odds (1:0.5:0.5), which keeps the odds of car, bus at (1:1), but the odds of car, blue bus have now fallen to (1:0.5). While this may seem like an overly specific problem to invent, there may be many cases where an alternative may be similar enough across individuals to induce the "bus choice apathy" seen in this instance.

Previously, much work had been done with path-based RCMs. Here, the set of possible alternatives is the set of paths in the network, but the goal remains the same; we wish to understand the underlying reasons behind observed behaviors. Examples of these types of

models include the C-Logit from Cascetta et al. (1996), Path Size Logit from Ben-Akiva and Bierlaire (2000), and the Cross Nested Logit from Vovsha (1997), all of which are discussed further in Prato (2009). Many of these models make use of certain variables to represent a correlation factor between alternatives to avoid the IIA which typically does not hold in route choice modeling. For example, the C-Logit model defines variables CF, representing the commonality factor, discussed in Cascetta et al. (1996), while the Path Size Logit uses path size. There are differences and similarities in each of these types of approaches, but all of these are attempts at solving the same problem: within a road network, of all the possible paths from and origin to a destination, many of the paths are going to contain many of the same links and thus share much of their properties.

A common issue faced in path-based models is choosing which of the possible alternatives to include in the selection of the choice set. With these path-based methods, given that the total number of possible actions can quickly grow out of hand, enumerating all possible paths can become difficult if not impossible. One workaround suggests choosing a subset of all possible paths that will be considered, where all other paths are assumed to have zero probability. This set of paths chosen to consider is known as the choice set, and is commonly denoted with C_n. The choice set presents several advantages, namely reduced computational time, reduced complexity, and more tractable solutions to the modeling problem. However, with any simplification comes issues, and choice set selection is not immune to this problem. An obvious issue is that an individual would prefer one of the actions not included within the choice set. As discussed in Wasi and Keane (2012), it is possible for randomly sampled choice sets to lead to consistent estimation of the parameters, yet achieving randomly sampled choice sets proves problematic when it is required to contain a specific alternative (such as an observed choice made by an individual in your data). However, there is a rich literature about the generation of choice sets, and for further discussion of the vast variety of choice set generation approaches, we direct the reader to the survey of Bekhor et al. (2006).

A second type of RCMs are the link-based RCMs, also known as recursive RCMs. The recursive term refers to how these RCMs view the paths traversed by individuals. Here, a path consists of a series of arcs between nodes, and the state space is made up from these arcs, rather than from all possible sequences of arcs from an origin to a destination. As described by Rust (1994), the problem solved in the link-based models is the estimation of the MDP parameters. The ε_a term is utilized once again to account for the noise intrinsic to the data, which allows us to formulate our estimation for the MDP's utility function as $u(a|s) = v(a|s) + \mu\varepsilon_a$, with s being the current state and a being a link alternative. This μ value is the scale associated with the error terms, allowing us to increase or decrease the influence of the assumed noise.

2.2. Shortest Path and Inverse Shortest Path Problems

Now, we introduce shortest path problems. The route choice problems we discuss are instances of shortest path problems, specifically the inverse shortest path problem, as this forms the forward problem for our optimization. We start by introducing the definitions and notation for the graphs we describe, followed by a description of various shortest path problems. We build successive problems on previous ones, starting with deterministic shortest path problems, and ending with inverse shortest path problems.

In a shortest path problem, we have a graph G composed of a tuple $(\mathcal{A},\mathcal{V})$, with $\mathcal{V}$ the set of all nodes in our graph G, and $\mathcal{A}$ the set of all links (arcs) between nodes[1]. For each node, we define a subset $A(i)$ to be the set of all adjacent links to node i, and for each link $a_{i,j} \in \mathcal{A}$, $a_{i,j}$ departs node i and arrives at node j. Furthermore, each link $a_{i,j}$ has an associated cost to its traversal, denoted $c_{i,j}$. This cost is a function $c : \mathcal{A} \to \mathbb{R}$. Here, we also define a path, which is a sequence of links where each subsequent link departs from the node the prior link arrived at, forming an unbroken walk along the graph G. The cost of this path is the sum of all link costs associated with each link chosen in the path. The goal of a shortest path problem is typically to minimize the cost of a path between a certain origin node and a certain destination node, though it is not uncommon for techniques to solve simultaneously for one origin node to all possible destination nodes, or vice versa.

The simplest of the shortest path problems are the deterministic shortest path (DSP) problems in which there is no uncertainty about where an action will lead. The primary hurdle for solving shortest path problems comes in the form of the large number of possible choices. Even for relatively small networks, the number of possible paths between two nodes is beyond feasible enumeration methods. However, as the paths are built of individual choices at each link for which departing link to take, many methods utilize Dynamic Programming (DP) methods to more efficiently tackle the problem. DP was introduced by Bellman to solve sequential problems where the outcome of preceding steps may be used to "guide the course of future ones" (Bellman, 1954). As an iterative method, it builds on previous values to minimize a cost over time steps. As we can decompose the shortest path problem this way, we can rely on the Bellman principle of optimality (Bellman, 1954) which states that optimal sequences are composed of optimal sub-sequences. Building on the principle of optimality, we use the Bellman Equation to represent the cost of a state as $C(k_i) = \min_{a_{i,j} \in \mathcal{A}(k)}(c_{i,j} + C(j))$, where the optimal cost of link i is the minimum sum between the link leaving i and the cost of the tail node it will arrive at. We can solve the DSP problem by assigning $\mathcal{V}$ to be our state set and $\mathcal{A}$ to be our action set. In order to use this DP formulation for one state in one time step, we define a recursive equation to solve for the optimal path from node i to

[1]Note that this set of vertices V differs from the definitions of value functions discussed, which we also have denoted $V(s)$. Here, the notation for the set of vertices is standard.

destination node d, as well as its related cost

$$C(k_i) = \begin{cases} \min_{a_{i,j} \in \mathcal{A}(i)} \left(c_{i,j} + C(j)\right) & i \neq d \\ 0 & i = d. \end{cases}$$

As mentioned in Zimmermann and Frejinger (2020), graphs, and therefore road networks, may have cycles, meaning unbounded recursion would result in infinite costs, or negative cycles may result in negative infinite costs. However, as we have assumed to be working with a connected graph, we can make the assumption that all nodes are reachable in at most $|V| - 1$ links. This is the approach taken by Bellman (1958), and we have an algorithm to compute not only the solution, but also the shortest paths between any node i and the destination node d in $0 < n \leq N$ steps, where $N = |V| - 1$:

$$\begin{cases} C_n(i) = 0 & \forall i | a_{i,d} \in \ \mathcal{A}(i) \\ C_n(i) = \min_{a_{i,j} \in \mathcal{A}(i)} \left(c_{i,j} + C_{n+1}(j)\right) & \forall i \in \mathcal{V}, n = 1, 2, \ldots, N-1. \end{cases}$$

The Stochastic Shortest Path (SSP) problem is similar to the DSP problem, with the difference between the two found in the transition probabilities between two states. In the DSP, we had that any action taking a link would result in arriving at the tail node for the link, i.e., choosing to take link $a_{i,j}$ from node i would take us to tail node j with probability 1. In SSP problems, these probabilities can take on any value in [0,1], not just the elements in {0,1}. In order to denote this change, we define $p(j|i, a_{i,j})$ to be the probability of ending up in state j having been in state i and taking action $a_{i,j}$. Luckily, we are still able to utilize the Bellman equation to write out a recursive value and the algorithm to solve for solutions. However, it is important to understand that with what has been defined, we cannot predict the best path, as each walk along the path may not walk through the same states even with the same actions taken, as we have a stochastic problem. Thus, our goal is to learn a policy π that will best traverse from origin to destination nodes in expectation. Our value function now takes the recursive formulation

$$V(i) = \min_{a_{i,j} \in \mathcal{A}(i)} \left(c_{i,a_{i,j}} + \sum_{j \in S} p(j|i, a_{i,j}) V(j) \right),$$

where we use the notation V instead of C to distinguish the stochastic case.

The inverse shortest path problem is simply an extension of the shortest path problem, where the cost function is unknown. Burton and Toint (1992) describe this problem and propose their solution. In their paper, they discuss the range of possible cost functions, from linear to decidedly non-linear functions. The goal of this line of work is to recover this unknown cost function, which then allows for novel predictions as we can estimate the decisions an expert would take, enabling us to evaluate link flows, traffic patterns, and route choice. In their paper, however, Burton and Toint (1992) sought to find the direct link

costs $c_{i,j}$ for all links $a_{i,j}$, instead of assuming any parameterized form of the cost function. This assumes that the modeler has access to the true costs of the links, something that is often unavailable in route choice modeling. The assumption holds in the problem considered in Burton and Toint (1992) as seismic topology and the propagation of seismic waves are known to follow the shortest path and observations of when these waves arrive at monitoring stations allow seismologists to compare their estimates for the geologic composition to the recorded values. Without the need for a parameterized cost function, the work in Burton and Toint (1992) and the following papers used the l_2 norm, or least squares norm, to impose structure on the minimization problem, leading to a quadratic programming formulation.

Route choice modeling is linked to inverse shortest path problems as both aim to infer values of unknown parameters in the objective function. In the case of RUM models, these are parameters in the utility functions, and in the case of inverse shortest paths, they are cost parameters. Making this link is useful as shortest path problems are not solved by path enumeration. (Recall that defining choice sets of paths is a key drawback of path-based RCM.) Instead the problem can be effectively solved by means of dynamic programming. However, RCMs are not deterministic. We therefore introduce Markov Decision Processes in the following section before discussing inverse stochastic problems in the section on inverse reinforcement learning.

2.3. Markov Decision Processes

Here, we introduce Markov Decision Processes (MDPs) that can be used to formulate sequential decision-making problems subject to uncertainty. One of the primary benefits of utilizing MDPs is that it provides a method to deal with problems that rely on future choices or decisions, as well as dealing well with uncertainty within the system. They are able to model decisions that impact future decisions through the passage of factors like time, states, or other dynamic variables. MDPs are composed of four parts. The first part is the state, as well as all the information surrounding the possible states. The state is usually denoted s_i, representing the i^{th} state, yet could also be expressed as s_t, which typically relates to the state at a time t. The information contained in s can be the physical state within the system, any information needed for the computation of the cost/reward function, and any other information necessary for making a decision. The second component is the action or decision, denoted as a_t. These actions can take on many values, be it binary, as it may be in situations such as stopping problems, discrete, where the action can take any value from a discrete set of values, or continuous, be it vectors or scalars. Other possibilities include categorical values, which can be represented as discrete actions, or vector-valued decisions. Many times, actions will be expressed as $a_t = A^{\pi}(s_t)$, where A is the total possible set of actions, and π is the policy that generates the action a_t from the set of A when given s_t. The

third part is the transition function P, describing how an actor moves between two states. This transition function is known by many other names, such as system model, plant model, or the transfer function. The transition function may be deterministic, where taking an action leads to the same outcome any time the action is taken from the state, or stochastic, where taking the same action from the same state may lead to alternative future states. The final piece of a MDP is the objective function, which may be a cost function or a reward function, which are simply negatives of each other as cost can be viewed as a negative reward. The objective function can be set to maximize cumulative rewards, where the rewards for all actions taken in a given span of time are summed together to give us a value of how good our decision were, or it can be set to maximize the final reward, where the reward at the terminus of our decision-making process is the value we wish to maximize. Lastly, we introduce the Markov property, where the current state for an agent is *only* dependant on the previous state, not the entire history of states until the present. This gives us that $p(s_t \mid s_{t-1}, s_{t-2}, \ldots, s_1) = p(s_t \mid s_{t-1})$. Using this framework, we can build out many different systems of sequential decision models that may be more beneficial to the problem we wish to model.

Now that we have provided descriptions of how we can frame networks for use with sequential decision-making, we can apply the discrete choice methods and introduce Dynamic Discrete Choice. Introduced by Rust (1987), Dynamic Discrete Choice (DDC) is a common method of modeling individuals choice making in an environment. A core tenet of DDC modeling is that agents will make decisions based on their current value of utility, and that this valuation of utility can change between states. This framework allows agents to model taking actions in order to maximize their utility through later states. An agent's choice of an action at a given state can be modeled as the maximum over an expectation,

$$E\left[\sum_{i=0}^{T} \gamma^i u(a_i, s_i) | a_0, s_0\right].$$

γ (sometimes represented as β in the literature) represents the discount factor, which weights utilities based on their temporal distance from our current context. This ensures that decisions far into the future will have less impact on the decisions much closer to our current state. This formulation of choosing actions with future utilities allows us to frame the problem in a Dynamic Programming framework.

One of the main goals of DDC modeling is to understand how changes in the system would impact choices made by individuals; for example, Rust's example of replacing bus engines seeks to learn how an increase in bus engine mileage impacts the value to the operator in their evaluation of rising operating costs compared to the cost of engine replacement. Solving the DDC problem involves utilizing the Nested Fixed Point (NFXP) algorithm, proposed by Rust

(1987), which is a gradient iterative search, combining the BHHH method to solve maximum likelihood equations, and value iteration to solve the dynamic programming problem.

With our DDC models, we seek to find parameters that maximize the action probabilities of the observed trajectories from our data. These trajectories, the set of which are denoted ζ, are composed of a sequence of $\zeta_n = \{s_0, a_0, s_1, a_1, \ldots, s_T, a_T\}$, containing the full set of states and actions taken from the origin to the destination. In order to accomplish this, we maximize the likelihood function, and thus, maximize the log-likelihood as well. It is common to use the log-likelihood instead of the likelihood as with many of models using the exponential family, transforming into the log space simplifies the computations by both reducing the exponential terms to linear ones and changes large products into more manageable sums like below,

$$L(\theta, \zeta) = \prod_{\zeta_i \in \zeta} \prod_{(s,a) \in \zeta_i} p(a|s)$$
$$LL(\theta, \zeta) = \sum_{\zeta_i \in \zeta} \sum_{(s,a) \in \zeta_i} ln(p(a|s)).$$

The parameter θ^* that will give the behavior we want is found at

$$\theta^* = \max_{\theta} LL(\theta, \zeta).$$

This θ^* is known as the maximum likelihood estimate and this estimation method is how we infer our parameters for our models from the data we have.

2.4. Inverse Reinforcement Learning

Reinforcement learning as a field aims to find a policy π that maps from the state space to a distribution over the action space which maximizes a reward in a given environment. In order to evaluate how well the policy maximizes this expected reward, we utilize a value function derived from previously mentioned Bellman Equation (Bellman, 1954). Denoted $V(s)$, it describes the value of being at state s, including the expected rewards for future states from the current state. Assuming stationarity (that is, the reward and state features are independent of time), we can write it as,

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left[R(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s') \right].$$

We make this assumption assuming an infinite horizon (classic in stochastic shortest path problems (Bertsekas and Tsitsiklis, 1991)). However, for finite-horizon problems, stationarity is not necessary. The infinite nature of infinite horizon problems leads to obvious difficulties in non-stationary formulations, as there can be vast differences in time steps in a single trajectory, leading to unnecessarily complicated relationships between the rewards. Another common equation used in reinforcement learning is the Q-function (also known sometimes

as the "state-action value") which is similar to the value function, except it measures the expected reward for a state-action pair. The Q-function is given by

$$Q_\pi(s,a) = R(s,a) + \sum_{s' \in \mathcal{S}} p(s'|s,a) V_\pi(s').$$

Note that the Q-function is the inner bracket value in the value function, so that the value function can be written as

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a).$$

Inverse Reinforcement Learning, first introduced by Russell (1998), sought to bring the successes of Reinforcement Learning to systems with partial knowledge. Much of the work done at the time relied on fully observable MDPs where the underlying state and all of its corresponding information is accessible to the modeler. In order to solve Partially Observable MDPs (POMDPs), where we are unable to observe the underlying state, we must rely instead on an observed value, which may be missing information when compared to the underlying state. In Inverse Reinforcement Learning, we have this POMDP without the given reward, R, that we would normally have with a MDP, but we are given a set of observations, ζ (Russell, 1998). These trajectories ζ are sequences of states and actions, such that $\zeta_i = \{(s_0, a_0), (s_1,a_1), \ldots, (s_T,a_T)\}$, which were generated under a policy π_{expert}. The IRL problem is that we do not know what π_{expert} is, or what weights it assigns to various features in the network. This problem of inferring parameters to compute this π_{expert} was first posed as a possible area of future research in the original extended abstract by Russell (1998). In it, Russel proposed using maximum likelihood estimation to solve this.

As we are unable to observe the underlying state, a common approach used to estimate the performance of a given policy π is known as feature matching (Abbeel and Ng, 2004). From their paper, we define a feature vector as $\mathbf{f} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$, a function that maps our state and action space to some d-dimensional representation. Within the route choice problem, this feature value representation commonly involves travel features such as link length, travel time, and speed limits, but also can include other features such as whether or not the link will result in a turn, the type of turn, or the type of road (highway, surface road, bus lane, etc.). With the feature function, we can define the feature vector $\mathbf{f}_{s,a}$ as the output of our feature function for the state-action pair (s,a). With this, Abbeel and Ng (2004) define the empirical feature count to be the average over the sum of feature vectors of the observed trajectories, that is, $\mathbf{f}_\zeta = \frac{1}{|\zeta|} \sum_{\zeta_i \in \zeta} \mathbf{f}_{\zeta_i}$, where $\mathbf{f}_{\zeta_i} = \sum_{(s,a) \in \zeta_i} \mathbf{f}_{s,a}$. As explained in Abbeel and Ng (2004), in order to match this empirical feature count, our formal goal is to find $\hat{\pi}$ such that $\mathbb{E}\left[\mathbf{f}_{\mathcal{S},\mathcal{A}} | \hat{\pi}\right] = \mathbf{f}_\zeta$, and thus match, in expectation, the actions taken in the observations.

Most IRL models use an approach known as Value Iteration (VI) in order to compute the optimal policy for a given MDP. Value Iteration is an iterative process where the estimation for the values of states is incrementally increased until the values converge. The process starts with randomly initialized values for each state, then loops until the values for the states converge. In each loop, the Q-value for each state-action pair is computed as $Q(s,a) = \sum_{s' \in S} p(s'|s,a)(r(s,a) + \gamma V(s'))$, and the new value for state s is given to be $V(s) = \max_a Q(s,a)$. Under certain assumptions, this process is guaranteed to converge to the optimal values, however convergence can be slow.

While these algorithms are used to tackle the RCM problem, they also served as possible solutions for other types of problems as well. In the Maximum Causal Entropy paper by Ziebart (2010), the algorithm is tested on a host of different problems. Some of them include helicopter control, where the model needed to learn how to hover about an origin point from a set of sub-optimal trajectories, a pursuit-evasion scenario, where the model was tasked with learning both chasing and evasion strategies for agents in the environment, and lastly a vehicular diagnostic and repair problem, where the model was tasked with choosing which components to evaluate in order to minimize the cost to the customer while still resolving the problem experienced by said customer. Previous work done in the field prior to the Ziebart Maximum Entropy model includes Apprenticeship Learning by Abbeel and Ng (2004), where they applied their algorithm to both a gridworld version of the RCM problem, as well as learning policies for driving vehicles, where they sampled driving styles from various expert policies and were able to successfully train their models to recreate the policies they observed.

Chapter 3

A Unified View of Existing Models and Algorithms

3.1. Notation

The purpose of this section is to define a common notation to use with both the DDC models and the IRL models, to facilitate comparison and mutual understanding of the algorithms. Recall that the structure of route choice can be modeled using a graph network, where our graph G is defined as $G = (\mathcal{A},\mathcal{V})$, where $\mathcal{A}$ is the set of all links in the graph and $\mathcal{V}$ is the set of the nodes in the graph, with the set $\mathcal{A}(k)$ to be the set of all links leaving the end of link k. With this graph structure, we can cast the route choice problem using the general framework of an MDP. As MDPs are defined by their states, actions, and transition probabilities between any two states, we define these variables as follows: the set of states, $\mathcal{S}$, is the set $\mathcal{S} = \{1,2,\ldots,|\mathcal{A}|\}$, our set of actions happens to be the same set $\mathcal{A}$, as the approach we take is link-based, not node-based. Lastly, the transition probabilities $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ map from our current state, action taken at the current state, and next state to a real number in the range of [0,1]. In practice, we assume our MDP's state transitions to be deterministic, where an attempt made by a driver to turn from one street to another street will not land them on a third street and will incur a known and deterministic cost in the form of travel time. The transition probabilities $p : \mathcal{S} \times \mathcal{A} \to \{0,1\}$ are then degenerate. In this case, it becomes a sequential decision-making problem rather than a MDP. Nevertheless, this abuse in terminology is common in reinforcement learning. Therefore, we continue to refer to MDP even if the state transition probabilities are degenerate.

We are also provided a set of observed trajectories, ζ, where for each ζ_i, $\zeta_i = \{k_0, k_1, k_2, \ldots, k_d\}$, and each k is a link chosen by the expert, which satisfies $k_{i+1} \in \mathcal{A}(k_i)$ for all $i < d$. These expert trajectories are assumed to have been created with

an unknown reward or cost function in mind [1]. This reward function, $R(k, a)$ represents the reward of taking link $a \in \mathcal{A}(k)$ at the end of link k. In order to estimate this reward function from the observations, our models are linear-in-parameter, with $\theta \in \mathbb{R}^f$, weighting the elements of the state features, denoted by $\mathbf{f}_s \in \mathbb{R}^f$, where f is the number of features which compose our feature matrix $\mathbf{F} \in \mathbb{R}^{f \times |\mathcal{A}|}$. In order for our models to estimate the reward of the trajectories, we let $R(\zeta_i) = \sum_{s_j \in \zeta_i} \theta^T F_{s_j}$. As mentioned in the previous sections, the reward is computed here with $r(s,a) = \theta^T \mathbf{F}_s$. This differs from the instantaneous utility as in many models, we add in an $\epsilon_n(a)$ term to represent the effect of any unobserved variables that may have affected the decisions of an actor n when considering link choice a. Since we are working with a multinomial logit model, these epsilon terms are i.i.d extreme value type I random variables with zero mean. However, this instantaneous utility is commonly calculated as the deterministic utility, our reward, plus this epsilon error term.

3.2. Recursive Route Choice Models

DDC modeling has a wide variety of applications, certainly not limited to route choice modeling. The technique has seen use in such diverse applications from estimating resource selection for wildlife, such as modeling elk summer bed selection (Cooper and Millspaugh, 1999) to modeling consumer demand in marketing (Chintagunta and Nair, 2011). This shows that the DDC framework can be applicable in many situations and thus, showing any overlap between the literature and methods of DDC and IRL may provide additional opportunities for those in unrelated fields to utilize novel algorithms for their own research.

Of these DDC models applied to the RCM problem is the Recursive Logit (RL) model. The RL model Fosgerau et al. (2013) gives probabilities of choosing an action a at state s according to the formula

$$P^d(a|s;\theta) = \frac{e^{\frac{1}{\mu} r(a|s;\theta) + V^d(a|\theta)}}{\sum_{a' \in \mathcal{A}(s)} e^{\frac{1}{\mu} r(a'|s;\theta) + V^d(a'|\theta)}}.$$

The recursive logit model is specific to a given destination, and needs to be solved for each destination in the data, hence the superscipts of a specific d. Let n denote the agent taking the route, that we assume to be traveling to some destination node. We extend the network by adding a new dummy link d which leaves the destination node and has the property $v(d|s) = 0$, for all s with $d \in A(s)$ Fosgerau et al. (2013). For the model, when we write the Bellman Equation for a given destination, we have

$$V^d(s) = E\left[\max_{a \in A(s)} r(a|s) + V^d(a) + \mu\epsilon(a)\right]. \tag{3.2.1}$$

[1] A reward function is simply the negative of a cost function, so for simplicity, we will only refer to reward functions henceforth.

We can also write the exponential of this value function as

$$e^{\frac{1}{\mu}V(s)} = \begin{cases} \sum_{a \in A} \delta(a|s) e^{\frac{1}{\mu} r(a|s) + V(a)} & s \in A \\ 1 & s = d. \end{cases}$$

By formulating the exponential of the value function in this manner, we can formulate our value function as the solution to a system of linear equations. We define a new destination specific matrix M^d with values $m_{s,a}$ defined as

$$m_{s,a} = \begin{cases} \delta(a|s) e^{\frac{1}{\mu} r(a|s)} & s \in A \\ 0 & s = d \end{cases}$$

where $\delta(a|s)$ is defined to be 1 when $a \in A(s)$ and 0 otherwise. Furthermore, define a vector z as a $|A| \times 1$ vector defined by $z_s = e^{\frac{1}{\mu}V(s)}$, and vector b a $|A| \times 1$ vector which has zeros in the s^{th} position for all k except our destination, which has a value of 1, we can solve a system of linear equations to find the values of the links. This system is expressible as $z^d = M^d z^d + b^d$.

Mai et al. (2018) provided a method to speed up the computation for the original recursive logit model, by utilizing matrix decomposition methods to pare down the number of linear systems which need to be solved. This also allows for the utilization of mixed recursive logit models to model correlated random terms.

The decomposition method proposed by Mai et al. (2018) takes advantage of the structure of these systems of linear equations by rewriting the learning algorithm using matrix notation for quickly solving the problem. We define a new matrix M^0, where $m^0_{s,a} = m^d_{s,a}$ if s,$a \in A$, and 0 otherwise. Define another matrix, U^d of size $|A| + 1$ by $|A| + 1$, with d a destination sink node, defined by $u^d_{s,a} = \delta(d|s) e^{\frac{1}{\mu} r(d|s)}$ if $a = d$, and 0 otherwise. Now, for each of the absorbing destination nodes $d \in D$, we have that $M^d = M^0 + U^d$. Mai et al. (2018) show that z^d is expressible as $M^0 z^d + t^d + b^d$, where t^d is defined to be the last column of U^d.

By defining two new matrices, B, of size $|A| + 1$ by $|D|$, with the d^{th} column being the column vector $t^d + b^d$, and Z, also of size $|A| + 1$ by $|D|$, with the d^{th} column being z^d, we can rewrite all the systems of equations to be $Z = M^0 Z + B$. This is equivalent to $(I - M^0)^{-1} Z = B$, which has a solution if and only if $I - M^0$ is invertible, the conditions of which are discussed in Fosgerau et al. (2013).

From this point, we need to chose a nonlinear optimization method to find our optimal solutions. There are a broad range of nonlinear optimizers (Nocedal and Wright, 2006), and the BFGS algorithm that we use in this thesis is just one of many that could be chosen. By using the BFGS algorithm, we can solve for the optimal parameters with the Jacobian and the Hessian matrices. For the Jacobian, we get that the gradient of Z with respect to the

parameter θ_i is

$$\frac{\partial Z}{\partial \theta_i} = (I - M^0)^{-1} \frac{\partial M^0}{\partial \theta_i} Z + (I - M^0)^{-1} \frac{\partial B}{\partial \theta_i}$$

and the Hessian is

$$\frac{\partial^2 Z}{\partial \theta_i \theta_j} = (I - M^0)^{-1} \left(\frac{\partial^2 M^0}{\partial \theta_i \theta_j} Z + \frac{\partial M^0}{\partial \theta_i} \frac{\partial Z}{\partial \theta_j} + \frac{\partial M^0}{\partial \theta_j} \frac{\partial Z}{\partial \theta_i} + \frac{\partial^2 B}{\partial \theta_i \theta_j} \right).$$

As mentioned, one issue with the Recursive Logit model is the IIA property. This is due to how the value function is formulated. Due to this, Mai et al. (2015) introduce a new approach which relaxes this property, the Nested Recursive Logit (NRL) model. The key to the NRL approach is link-specific scale parameters, μ_s. While in the RL model, the instantaneous utility is given by $u(a|s) = r(a|s) + \mu\epsilon_a$, we now have $u(a|s) = r(a|s) + \mu_s\epsilon_a$. This scale parameter is strictly positive, and differs from the RL model in that the RL model assumes that for all s, $\mu_s = \mu$. While RL model specifies its value function in (3.2.1), the NRL value function is given by

$$\frac{1}{\mu_s} V^d(s) = \ln \left(\sum_{a \in \mathcal{A}(s)} e^{\frac{1}{\mu_s}(r(a|s) + V^d(a))} \right).$$

Similarly to the RL approach, we can rewrite the exponential of this value function to be

$$e^{\frac{1}{\mu} V^d(s)} = \begin{cases} \sum_{a \in A} \delta(a|s) e^{\frac{r(a|s) + V^d(a)}{\mu_s}} & s \in A \\ 1 & s = d. \end{cases}$$

Since these scale parameters are link-specific as mentioned above, we relax the IIA property, as the scales do not cancel out like in the RL model.

Literature surrounding route choice modeling has utilized a variety of methodologies as well as measuring different metrics to evaluate these models. In Fosgerau et al. (2013), they utilize a feature space of only 4 features, travel time, left turn, a link constant, and finally a u-turn.

3.3. Maximum Entropy Models

Of the IRL models discussed in this thesis, the Maximum Entropy (ME) model (Ziebart et al., 2008) is the simplest as well as the foundation for the models discussed later in this section. In order to understand this algorithm, it is important to discuss the concept of entropy. Taken from Information Theory, entropy is a measure of the information or surprise there is in a system (Shannon, 1948), representing the number of bits required to represent the average outcomes. Information of an event x occurring with probability $p(x)$ in this context is defined to be $I(x) = -\log(p(x))$. It is evident that an event with a high probability close to 1 would have very little information associated with it, while a rare event would have a large value of information. When taken with the idea of entropy, we see that by

maximizing the entropy across our probability distribution, each event will provide us more information than if we have low entropy across our distribution and require less assumptions of the distribution. By maximizing the entropy, we ensure that the model is optimizing the feature matching as discussed in Section 2.4, without assigning additional preferences to links other than their rewards. Out of this idea, we get a probability distribution for a trajectory ζ_i with parameters θ of

$$P(\zeta_i|\theta) = \frac{1}{Z(\theta)} \exp \mathbf{f}_{\zeta_i}$$

where $Z(\theta)$ is the partition function, which represents the likelihood of all trajectories. Ziebart et al. (2008) (i) define a probabilistic model according to a "max entropy principle" and (ii) maximize the cross-entropy of their model with the empirical distribution of the observed data, with respect to its parameters. We note that (ii) is equivalent to pseudo maximum likelihood and with minimization of KL divergence with the empirical distribution of the data.

With entropy and the partition functions explained, we move to the algorithm proper. The ME algorithm is composed of a backward pass and a forward pass akin to the value iteration approach discussed previously. In the backward pass, we compute the probability mass for each state and action, effectively computing the partition function. This is accomplished by recursively "backing up" from each terminal state, computing the probabilities for adjacent (lines 4,5 of Algorithm 1). In line 8, we compute the local probabilities by dividing our probability masses by their respective partition function value. In the forward pass, we compute the expected state visitation frequencies using these local probabilities (line 11), taking their sum to give us our total state visitation frequencies under θ in line 12. Line 13 is our update state, as we adjust our parameter vector by the resulting difference between the observed feature expectation and the computed feature count.

Building on the ME model, Ziebart (2010) introduced the Maximum Causal Entropy (MCE) model. The MCE model extends the ME approach of maximizing the entropy to conditional probability distributions where there is a conditionally causal relationship between the actions and states. Under a trajectory ζ of length T with actions and states A^ζ and S^ζ respectively, we can express this conditionally causal relationship, $P(A^\zeta||S^\zeta)$, as

$$\begin{aligned} P(\zeta) &= P(A^\zeta||S^\zeta) \\ &= \prod_{t=1}^{T} p(A_t^\zeta|A_{1:t-1}^\zeta, S_{1:t}^\zeta). \end{aligned}$$

Note that causally conditional probabilities are denoted with the double vertical line, as opposed to the single vertical line with normal conditional probabilities.

The motivation behind this addition was that it provides a certain directionality for the information. For example in the RCM problem, if equal amounts of individuals were observed

Algorithm 1: Maximum Entropy IRL Ziebart et al. (2008)

Data: Observed trajectories ζ
Result: θ parameters
Initialize θ
Set $\tilde{\mathbf{f}} = \frac{1}{|\zeta|} \sum_{\zeta_i} \mathbf{f}_{\zeta_i}$
for *each iteration until stop* **do**
  Set $Z_{s_i} = 1$
  for *N iterations* **do**
    $Z_{a_i,j} = \sum_k P(s_k|s_i, a_{i,j}) Z_{s_k} e^{\theta^T \mathbf{f}_{s_i}}$
    $Z_{s_i} = \sum_{a_{i,j}} Z_{a_{i,j}}$
  $P(a_{i,j}|s_i) = \frac{Z_{a_{i,j}}}{Z_{s_i}}$
  Set $D_{s_i,t}$ to initial state probabilities
  for *N iterations* **do**
    $D_{s_i,t+1} = \sum_{a_{i,j}} \sum_k D_{s_k,t} P(a_{i,j}|s_i) P(s_k|a_{i,j}, s_i)$
  $D_{s_i} = \sum_t D_{s_i,t}$
  $\theta = \theta + \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i}$

approaching the same intersection from two opposite directions only to continue down the same direction they had been driving, the traditional ME approach would state the two links they took should have equal probability when at that intersection. This example could lead to a prediction of a u-turn for an individual, which may not make sense if their destination was previously in front of them. However, with the conditioning on past states and actions, the MCE model would better predict that given the directionality of their route, they would continue on in their direction, and avoid taking an unnecessary u-turn or the like.

One of the drawbacks to the previous IRL approaches is that the reward function derived from the data may explain the trajectories observed by experts, whereas it has no concept of correlation between connected states in the system. This drawback was the inspiration behind the Generalized Maximum Causal Entropy (GMCE) algorithm (Mai et al., 2019), which sought to generalize the MCE model proposed by Ziebart (2010). Similarly to how the Nested Recursive Logit Algorithm from the previous section relaxed the IIA property, the GMCE algorithm does as well, for the same reasons. An example of the benefits of breaking this assumption in the IRL field is given in their paper (Figure 1 of Mai et al. (2019)), where classical maximum entropy approaches would assign equal weight to the three possible paths, even though 2 of the 3 paths share half of their links. By expanding upon the existing approach, their algorithm devalues these two paths as they are functionally equal, leading to a more intuitive solution when compared to the predecessors. To incorporate this idea into their algorithm, they utilize the same method of feature matching, however. Their features, $\mathbf{F}_s^{\mu}$, contain not only the same state features as would be expected in the maximum

entropy approach, but also some information on the "overlapping-level" Mai et al. (2019) of state s. This addition of a connectivity attribute can be viewed as this algorithms approach to the link size attribute proposed in Fosgerau et al. (2013).

The GMCE algorithm progresses in much the same way as the ME algorithm, with the computation of the probability masses (lines 13 and 14 of Algorithm 2) and state visitation frequencies (lines 15 and 16), denoted once again with Z and D respectively. For an in-depth definition of the function G, we refer the reader to the original paper by Mai et al. (2019). The function, $G(p|s)\colon \mathbb{R}^+ \times \mathcal{S} \to \mathbb{R}^+$, is assumed to be positive and differentiable with respect to p, which is also positive, invertible such that $G(G^{-1}(p|s)|s) = p$, $\forall p \in \mathbb{R}^+$, and lastly, that there exists $\mu\colon \mathcal{S} \times \mathbb{R}^+$ such that $\frac{\partial \ln(G(p|s)}{\partial p} = \frac{\mu(s)}{p}$. These assumptions give us that there exists a mapping $v\colon \mathcal{S} \to \mathbb{R}^+$ such that $G(p|s) = e^{v(s)}p^{\mu(s)}$. From these, we have enough defined to define the algorithm.

Algorithm 2: GMCE LL and Gradient Computation Mai et al. (2019)

Data: Observed trajectories ζ
Result: θ parameters
1 Initialize θ
2 **for** $t = T,\ldots,1$ **do**
3 **if** $t = T$ **then**
4 **forall** a_t, s_t **do**
5 $Y_{a_t|s_t} = R(s_t, a_t)$
6 $U_{a_t|s_t} = \nabla_\theta R(s_t, a_t)$
7 $Z_{s_t} = 1,$ and $D_{s_t} = 0$
8 **else**
9 $Y_{a_t|s_t} = R(s_t, a_t) + \sum_{s_{t+1}} p(s_{t+1}|a_t, s_t) \ln G(Z_{s_{t+1}}|s_{t+1})$
10 $E_{s_t,s_{t+1}} = \frac{\partial G^{-1}(z|s_t)}{\partial z}|_{z=Z_{s_{t+1}}} \frac{D_{s_{t+1}}}{G(Z_{s_{t+1}}|s_{t+1})}$
11 $U_{a_t|s_t} = \nabla_\theta R(s_t, a_t) + \sum_{s_{t+1}} p(s_{t+1}|a_t, s_t) E_{s_t,s_{t+1}}$
12 **forall** $a_t \in \mathcal{A}^t, s_t \in \mathcal{S}$ **do**
13 $Z_{a_t|s_t} = G^{-1}(e^{Y_{a_t|s_t}})$
14 $Z_{s_t} = \sum_{a_t} Z_{a_t|s_t}$
15 $D_{a_t|s_t} = \frac{\partial G^{-1}(z|s_t)}{\partial z}|_{z=e^{Y_{a_t|s_t}}} e^{Y_{a_t|s_t}} U_{a_t|s_t} + \nabla_\theta G^{-1}(e^{Y_{a_t|s_t}}|s_t)$
16 $D_{s_t} = \sum_{a_t} D_{a_t|s_t}$
17 **for** *any observation* $(\tilde{s}_t, \tilde{a}_t)$ **do**
18 $\ln P(\tilde{a}_t|\tilde{s}_t) = Z_{\tilde{a}_t|\tilde{s}_t}/Z_{\tilde{s}_t}$
19 $\nabla_\theta \ln P(\tilde{a}_t|\tilde{s}_t) = D_{\tilde{a}_t|\tilde{s}_t}/Z_{\tilde{a}_t|\tilde{s}_t} - D_{\tilde{s}_t}/Z_{\tilde{s}_t}$

The Maximum Entropy algorithm proposed by Ziebart operated on a MDP modeled on a road network surrounding and including Pittsburg Pennsylvania, and observations generated from 25 taxi cab drivers across the span of 12 weeks. This generated around 13,000 distinct trips, covering more than 10,000 miles. In order to represent the paths, they modeled 22

distinct features, including the road type (highway roads through to low speed local roads), the indicated speed limit (low speed to high speeds), the number of lanes (one lane through many lanes), and the inter-road transitions (i.e. hard left, left, straight, right, hard right turns). Furthermore, they split their data into a 80-20 training-test sets, in order to test their data on data that appears novel to the model. In their measuring of the success of their algorithm, they use three metrics: path matching, 90% path matching, and the log probability of most likely paths. Both path matching and 90% path matching are metrics based on the percentage of overlapping route distance between what the model predicts for a given origin and destination, with path matching being the raw value, and 90% path matching representing the percentage of trips where the path matching value is over 90%.

The GMCE algorithm proposed by Mai et al. (2019) operated on an unnamed transport dataset consisting of 1,832 trajectories of taxi cab drivers on a road network with 7,288 links. Unlike the 22 features used to describe states in the Ziebart et al. (2008) example, Mai et al. (2019) used only four features. The first three were binary, representing if the action was a left turn, a u turn, or at an intersection. The fourth and final feature was the travel time between the connected links. Again, an 80-20 training-test split is created for the evaluation, and the metrics monitored are the same as Ziebart's Max Entropy example. While this is useful for the comparison they made between their algorithm and the algorithm it was generalized from, it represents a clear propagation of a blind spot in the IRL route choice modeling literature, as no other metrics were utilized.

3.4. Discussion

Both the econometric DDC models and the IRL models use functions to calculate probabilities based on values, utilities, or costs. The multinomial logit model with i.i.d. extreme value type 1 error terms gives a probability of $P(a_i|s) = \frac{e^{f(a_i,s)}}{\sum_{a_j \in \mathcal{A}(s)} e^{f(a_j,s)}}$, with function f representing the specific value function. To those in the world of machine learning, this is better known as the softmax function[2]. This exponential function is the exact framework for both the original Maximum Entropy (ME) algorithm and the Recursive Logit models, with different value functions. For ME, this value is $f(a_i, s) = Q(s, a_i)$, while for the IRL model, the utility is $f(a_i, s) = \frac{1}{\mu_s}(v(a|s) + V^d(a))$.

In addition, with both the GMCE and NRL approaches, the IIA assumption is relaxed. Thus, we pose that GMCE is to ME as NRL is to RL, providing us a wonderful opportunity to compare these approaches. As discussed, the reasoning behind this relaxation of the IIA assumption has long been discussed (McFadden, 1977), and can prove to be a useful assumption to drop.

[2]It should be noted that while the Ziebart (2010) paper describes using the softmax function for calculating probabilities, the function used is the Log-Sum-Exp function, which is distinct from the typical definition of softmax in ML.

3.5. Performance Metrics

A wide variety of performance metrics have been used in the literature, and as one of the goals of this thesis is to attempt a direct comparison between algorithms from the two fields, we attempt to take metrics from both strands of literature to align the comparison. One theme we found was that the work from econometrics measures the shape of the distribution around the estimated parameters, while the IRL methods lean towards measuring accuracy of the predictions when compared to the observed trajectories. This is natural as the maximum likelihood estimator has nice properties, namely consistency and efficiency. Thus, values such as the average and standard deviation of the estimated parameters are often reported in the papers originating from econometrics. This is not to say that there is no overlap between the two fields' commonly reported performance metrics: the most common metric is the log-likelihood on the data. This is a relatively simple metric to measure as with the maximum likelihood estimation, log-likelihood has already been computed as part of the estimation procedure.

Other metrics besides those that fit into the aforementioned categories have also been studied; in Mai et al. (2018), in addition to the log-likelihood, they also report computational performance. For example, the time taken in seconds for both gradient computation and estimation of their algorithm compared to similar algorithm that theirs is derived from.

Of the many choices for metrics to choose from, we take eight to focus on. One of these is the "path matching" percentage (%PM); this is the average percentage of the length of matching links between the observed trajectories and their respective generated trajectories, where 100% would be a perfect match between the observed and generated trajectories with each generated trajectory taking the exact same path as the observed trajectory it is being generated against. A similar metric that is often used alongside the path matching percentage is the "90% path matching" (90%PM), which represents the percentage of generated trajectories that share at least 90% of their component links with their observed trajectory counterparts. Perhaps the most common metric is the average log-likelihood of observed paths under the given model and its parameters on the holdout set. While we cannot always directly compare the log-likelihood values between two models, we report the values here as there are direct similarities between both the RL and NRL models. For the RL and NRL comparison of log-likelihood data, the RL model is functionally the NRL model with the scale parameters set to 1, whereas the NRL model learns the scale parameters. The difference in number of parameters can be one potential issue with this comparison, as the RL and NRL models differ in that the NRL model has additional scale parameters. Thus, we report the LL values with the understanding that they may be not directly comparable, but with the hope we can gain some insight in their individual performances on the data. We measure these values for both the train sets and the holdout test sets, measuring the

in-sample and out-of-sample performances respectively. By measuring both sets, we can form a better understanding on how our models perform on novel data relative to the data it has already seen. Lastly, we have multiple measures to understand the amount of time or compute needed to run the models. For example, the value time represents the time in seconds that it takes to compute the value function for all states in the network. While we measure the average time to train the model and the average time to evaluate the model, our models are written in two different programming languages (Matlab for the RL and NRL models and Python for the ME and GMCE models). Thus, we also report the average number of training iterations taken by each model. As this will be language independent, this should provide a level comparison between the models.

Finally, we note that we are comparing the inner algorithm in the parameter estimation, and we do not measure the impact of the outer algorithm on the estimation procedure. Different optimization algorithms may lead to differences in the number of iterations of the outer loop. However, we leave an exploration of the impact of various outer algorithms in the optimization as possible future work.

Chapter 4

Numerical Results

In this chapter we discuss our results as well as the issues which prevented us from computing results for all of our models. We discuss the performance of our models on a small, toy dataset, for which we were able to generate results, as well as discuss the results we were able to generate for a much larger dataset, consisting of GPS traces of cyclists in the US city of Portland. After the discussion of results, we explain the causes behind the issues with our results, both from the numerical side as well as the programming side.

4.1. Illustrative Example

We start by discussing the results of our models on a toy dataset, consisting of 19 total links (Figure 4.1). This dataset has previously been used in Zimmermann and Frejinger (2020) in their comparison between the path-based Logit model and the RL model, the former of which uses choice sets to enumerate the set of possible paths instead of using the link-based approach of the RL model. The two features used in this model are the travel times (denoted $TT(a)$), seen as the weights associated with the links in the figure, and a link constant ($LC(a)$), with a value of 1 for all links in the network. With this synthetic dataset, the observations are generated under the assumption that the true utility is given by

$$u(a) = (-2.0 * TT(a)) + (-0.01 * LC(a)) + \varepsilon(a).$$

Using this utility, 500 trajectories have been generated starting from node o and ending at node d. Of these, 400 are used for training, and 100 used for testing, an 80-20 split common to 5-fold cross-validation. One feature of the structure of the network is its acyclic nature, where once a node has been left, there is no route to return to it. While this acyclic feature is not necessary for any of the models we have discussed, it does limit the maximum length of a trajectory as well as speeds up convergence of the algorithms.

The testing for this network was done using a 2017 MacBook with a 3.5GHz processor. In our testing, we were able to achieve results for the RL, NRL, and ME model, but ran into

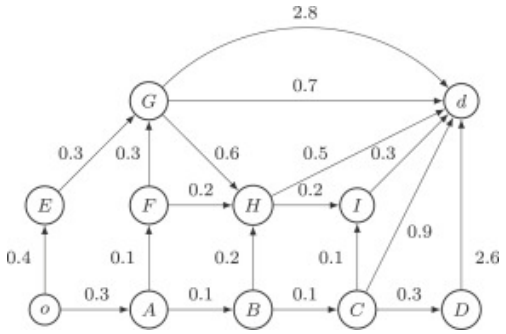

Fig. 4.1. Toy Dataset Network (Zimmermann and Frejinger, 2020)

Model	RL	NRL	ME
Log Likelihood on Train set	-2.309	-2.309	-2.309
Log Likelihood on Test set	-2.312	-2.312	-2.318
Nb. Train Iterations	8	8	55
Average Time Taken during Training (sec)	0.794	4.538	19.208
Average Time for Testing (sec)	0.0	0.0	0.0
Avg. Computation of Value Function (sec)	0.086	0.210	0.092

Table 4.1. Results on Toy Dataset

programming difficulties with the GMCE model. For the models we achieved results with, we see that the performance of the three are all very similar. We see the log-likelihoods of all the models on the train set to be equal, with only their testing log-likelihood differing, with the ME model having a lower testing log-likelihood by 0.006. The biggest difference between the models tested comes in their time taken, specifically with the ME model taking almost 5 times as long as the NRL model to complete the parameter estimation. However, the seconds-per-training-iteration of the ME model is 0.349 seconds, less than the NRL model's 0.567 seconds. This difference in iterations is traceable to the RL and NRL approach of solving the value function as a system of linear equations, whereas the ME model approximates the value function using the value iteration method.

The issues with the GMCE model arose both due to the lesser number of features, which our framework had not been set up to handle, as well as issues with the optimization library used (SciPy). These issues were unexpected, as even with changes to the code to account for the reduction of the feature space, numerical errors were raised by the optimization library used. The solution to the latter has thus far eluded us, but is believed to be a type disagreement within the computations of the value function.

4.2. Bike Route Choice Results

Next, we move to a much larger, real world dataset. We call this dataset the Portland Bike dataset, as it is composed of 42,000 links from the Oregonian city of Portland's road

network as well as 648 GPS traces of cyclists navigating an average of approximately 50 links each, with a maximum of 261 links traversed. Furthermore, as this is a real-world network, the graph is highly cyclic, especially in comparison to the toy network described previously. The attributes we will use for the instantaneous utilities of our models consist of the same attributes as in Fosgerau et al. (2013), namely link travel time, a left turn value $LT(a|s)$ with value 1 if the turn from link s to link a is between the degree value of 40 and 177, and u-turn value $UT(a|s)$ with value 1 if the turn from link s to link a has value greater than 177, and a link constant set to 1 for all links. With our models, the utility for links within this network is specified by

$$u(a|s) = \theta_{TT}TT(a) + \theta_{LT}LT(a|s) + \theta_{UT}UT(a|s) + \theta_{LC}LC(a) + \varepsilon_a.$$

For our experiments, we utilize cross-validation by splitting the observation data into 5 roughly equal-sized subsets, 3 sets of 130 and 2 of 129. To ensure fair testing, we split the data once, ensuring all models are run on the exact same train/test sets. This allows us to run 5 tests using completely unseen holdout sets of 20% of our total data each run, giving us a fair estimate of our performance metrics. For the path matching, we compute this value by generating trajectories with the same origin node and destination node that the corresponding trajectories utilize. Once we have the generated trajectories, we compare each trajectory's generated and observed data, summing the shared path length, and dividing by the number of trajectories generated. After computing the path matching percentage for each of the holdout sets, we take the average of the 5 runs to get our final reported value. For the testing, the models were run on m5.4xlarge Amazon Web Service (AWS) instances running Linux, as these provided us 64 GB of available memory.

Model	RL	NRL
Path Matching (avg %)	35.62	35.37
90% Path Matching (avg %)	7.86	7.20
Log Likelihood of Train Set (avg)	-29.145	-29.792
Log Likelihood of Test Set (avg)	-29.256	-29.534
Time to Train (avg sec.)	29.145	124,147
Number of Training Iterations (avg)	42	41.25
Time Test (avg sec.)	27.449	37.695
Value Function Time (avg sec.)	19.983	106.058

Table 4.2. Average of Cross-validation Results for the Portland Bike Dataset

Of the four models, we were able to generate results for only the RL and NRL models. However, there are important insights to be gleaned from the limited results. First, as will be discussed in the next section, the NRL model took significantly longer than the RL model, with the NRL taking approximately 34 hours compared to the RL models' 30 seconds. With that large increase in training time, the NRL model took a very similar number of iterations

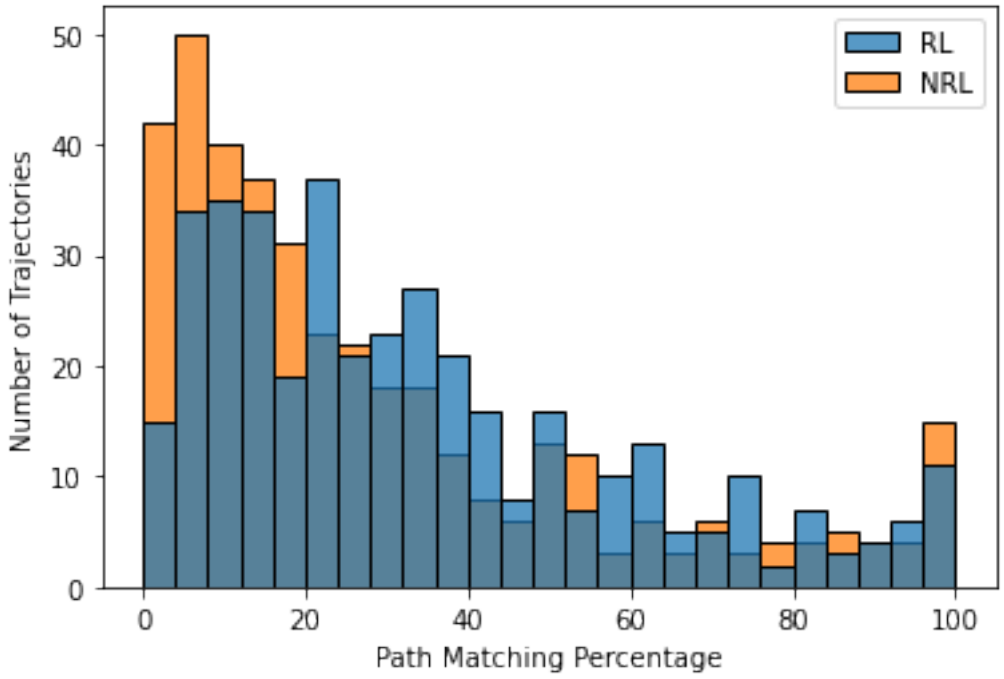

Fig. 4.2. Distribution of Shared Lengths on Portland Bike Dataset

to converge to a solution to the RL, and both solutions tended to be fairly equal. The RL model achieved slightly better log-likelihoods on both train and test sets. However, the difference is not significant (likelihood ratio of 0.556, which for the one degree of freedom between the models, gives a p-value of 0.455. The path matching performances are also similar.

In Figure 4.2, we can see the cumulative distribution of the shared proportions of length between both the predicted routes and the observed routes for origin-destination pairs, with the RL being represented in blue, NRL in orange, and their overlap having a dark blue color. From this, we can see that the NRL model had significantly more trajectories with less than 20% path matching, and only slightly more trajectories with over 95% path matching when compared to the RL model. Furthermore, the RL model had a more even distribution than the NRL model, showing that the predictions the RL model made after training were slightly more accurate in predicting the routes a novel driver would take through the network.

4.3. Numerical Issues and Programming Challenges

As there were many issues with the generation of results, both from the models which we were unable to generate results for as well as issues with the models for which results could be generated, we provide explanations for these issues as well as solutions we found. We split the insights gained into two categories: numerical issues and programming challenges. The reflections on numerical issues focus on problems we had with regards to the data or how

our algorithms interacted with the data. The reflections on programming challenges center around issues that we experienced when executing our programs, including unreasonable running times, memory overhead issues, and the computation of certain aspects of the data itself.

4.3.1. Numerical and Data Issues

Of the numerical issues experienced in this thesis, numerical stability was a repeated issue. When implementing the code for the Maximum Entropy IRL (Ziebart et al., 2008), we continually suffered from underflow issues in our value function computations. Due to the iterative nature of this computation (lines 6 and 7 of Algorithm 1), and the small values found when working with probabilities in general, it was common for our computed probability masses to vanish. This caused issues as in line 8 of the same algorithm, we divide by the sum of the probability masses assigned to a given state, and when underflow issues caused these vanishing probabilities, our policy computations would become undefined due to them dividing by zero.

To deal with this numerical stability issue, we elected to transfer the computation of the probabilities to the log-space, as was done in Ziebart et al. (2010b), with a few alterations. In the original derivation of the Maximum Entropy algorithm (Algorithm 1), the probability computation consists of iterating over the loop

$$Z_{a_{i,j}} = \sum_k P(s_k|s_i, a_{i,j}) Z_{s_k} e^{\theta^T \mathbf{f}_{s_i}}$$
$$Z_{s_i} = \sum_j Z_{a_{i,j}}.$$

Here, we make a simplification to this computation due to the deterministic nature of our state transitions. In the calculation for the action specific Z value, the only non-zero value of $P(s_k|s_i, a_{i,j})$ is when k is equal to j, so we can remove the sum in favor of computing the value directly. For simplicity, we drop the $P(s_k|s_i, a_{i,j})$ term as it will be equivalent to 1 with how the transition matrix is defined. This allows us to compute $Z_{a_{i,j}}$ as

$$Z_{a_{i,j}} = Z_{s_j} e^{\theta^T \mathbf{f}_{s_i}}.$$

With this computation reframed in this manner, transforming the calculations to the log space is trivial:

$$\begin{aligned}\log Z_{a_{i,j}} &= \log Z_{s_j} + \theta^T \mathbf{f}_{s_i} \\ &= \log Z_{s_j} + r(s_i, a_{i,j}).\end{aligned}$$

With these changes, the algorithm no longer experienced numerical stability issues in the value function computation. In addition, this formulation allowed us to compute the value function using matrix addition, speeding up the computation time of our algorithm. For

the matrix addition, we used the transition matrix induced by the original structure from the computation of $Z_{a_{i,j}}$ that relies on looping over all possible actions $a_{i,j}$ by element-wise multiplying the transition matrix by the transpose of our $\log Z_s$ values

$$\log Z_a = P \otimes \log Z_s^T + R.$$

The full algorithm we used can be seen in Algorithm 3. The transition to the log space for the probability mass computation is visible on lines 7 and 8. In addition to the numerical stability gained from this, many mathematical libraries have optimized "LogSumExp" methods, allowing us to compute line 8 efficiently (See NumPy, SciPy, Torch, Tensorflow for their various implementations of this function). Computing the probabilities for our policy on line 9 is equivalent to the calculation in the original algorithm, when shifted to the log space, where our division becomes subtraction. The other notable change comes in the computation of the state visitation frequencies, where we no longer store a history of past D_s values, and instead iterate on the same variable, with the transpose of the policy probabilities being element-wise multiplied by the vector result of adding the current value of D_s with the initial probability.

Algorithm 3: Our Numerically Stable Max Entropy IRL

Data: Observed trajectories ζ
Result: θ parameters
1 Initialize θ
2 Compute initial state probabilities p_0
3 Set $\tilde{\mathbf{f}} = \frac{1}{|\zeta|} \sum_{\zeta_i} \mathbf{f}_{\zeta_i}$
4 **for** *each iteration until stop* **do**
5 Set $Z_{s_i} = 1$
6 **for** *N iterations* **do**
7 $\log Z_a = P \otimes \log Z_s^T + R$
8 $\log Z_{s_i} = \log \sum_{j \in \mathcal{A}(s_i)} \exp Z_{a_{i,j}}$
9 $P_\pi(a_{i,j}|s_i) = \exp\left(Z_{a_{i,j}} - Z_{s_i}\right)$
10 **for** *N iterations* **do**
11 $D_s = P_\pi^T \otimes (D_s + p_0)$
12 $\theta = \theta + \tilde{\mathbf{f}} - \sum_{s_i} D_{s_i} \mathbf{f}_{s_i}$

In addition to the numerical instabilities, we also encountered issues with portions of the data. The primary dataset we sought to test on was the Portland Bike dataset, as the number of links was relatively small compared to the alternative dataset we had access to (42k in comparison to 1.5m). One issue we came across with this data was the previously parsed data and existing code given to us had discrepancies between the stated values and the values we observed. One example of this is in the turn data: as mentioned, we define turns as being link transitions within specific link angles (i.e. a left turn is only a left when

the angle between the links is between 40 and 177 degrees Mai et al. (2015)) while the existing code and data had angle requirements with a range of 150 degrees as opposed to the cited 137 degrees. This caused a perplexing issue where models were training towards notably different minima, when they should have been converging towards similar solutions, even though they had theoretically been training on the same data.

4.3.2. Programming Challenges

We split the programming challenges into two categories. The first is memory overhead issues. With such large networks, algorithms which required storage of many variables, or implementations which required many dense representations of the data, quickly surpassed that amount of available memory on the system. The second category is that of implementation issues and issues surrounding the integration of the data into the implementations.

With regards to the memory issues, we found both the NRL and the GMCE implementations had significant memory overheads. The required amount of memory was so significant that when we attempted testing on the AWS instance we had originally planned to use, the m5.xlarge, the 8 GB of memory were quickly allotted and the programs would crash during training. With an inability to train, we were unable to test the algorithms in the state which we received them. In response, we revisited the implementations to optimize their memory usages. Within both the NRL and GMCE implementations, we found that many of the computations kept large matrices loaded in memory for quicker access for future computations. However, while re-computing these matrices each time they were needed would be computationally more expensive, we found that by deallocating the variables, such as the intermediates for computing value functions or gradients, we could decrease the memory overhead by approximately 20 GB for the NRL implementation, and by over 100 GB for the GMCE implementation, when training with the Portland Bike dataset. This reduction in memory overhead for the NRL model allowed us to train the algorithm using the larger AWS instance we mentioned with 64 GB of memory, and the increase in time spent recomputing these somewhat frequently used variables may play a part in the vast difference in training time between the RL model and the NRL model. In addition to the removal of these stored variables, the GMCE code contained variable redundancies that further stretched the available memory. The original implementation contained hundreds of lines of code dedicated to repetitive computations, such as computing the gradient for each scale parameter. By replacing these large blocks of code with a combination of loops and pythonic list comprehension, the reassignment of variables that had been left behind reduced the memory usage by a factor directly proportional to the number of parameters used.

Even with the memory reduction for the GMCE model, the AWS instance we had selected struggled to run our training program. Thus, we reframed much of the data to consolidate

the information into sparse matrices. The difficulties came with the dimension of the data the GMCE implementation was using, with the various forms of transition matrix having the form $P\colon |\mathcal{S}| \times |\mathcal{A}| \times |\mathcal{S}| \to \mathbb{R}$. As mentioned, the deterministic nature of the state transitions allows us to simplify the construction of the transition matrix. Given what our transitions represent, taking the action $a_{i,j}$ would only be a non-zero value when going from state i to state j, leaving $|\mathcal{A}| - 1$ empty sets of the final $|\mathcal{S}|$ dimension. While n-dimensional sparse matrices are certainly possible to use, 2-dimensional sparse matrices are the most intuitive and flexible when it comes to matrix multiplication or manipulation, as well as having highly optimized libraries for their computations. Thus, we reduced our matrices to be of the form $|\mathcal{S}| \times |\mathcal{S}|$, with the probability of each $a_{i,j}$ being found on the i^{th} row and the j^{th} column of our P matrix.

With the reduction of redundant or non-critical code in conjunction with the replacement of dense matrices by their sparse counterparts, we were successful in starting the training for the GMCE implementation. While the exact amount was not critically studied, these changes achieved memory improvements of approximately 90% (6.5 GB of memory compared to 500 MB of memory when tested with a 7k link dataset). Additionally, with the code written in Python, we were able to achieve computational time decreases by moving repeated calculations to list-comprehension approaches. These changes made it possible to run the code on the AWS server architecture that we had agreed to use, as the dataset we tested on was approximately 40 times larger than this small dataset.

Similarly to the values reported in Zimmermann and Frejinger (2020), we found that the NRL code ran significantly slower than the RL code. This can be seen in our current results, with the training time taking an average of 36 hours when compared to the Decomposition method for the RL, which took 30 seconds. This made for incredibly slow testing and debugging, yet the GMCE model took even longer. While we were able to start the GMCE code, the first iteration took over 27 hours to complete. This kept us from successfully training the model, as even if the GMCE model converged within around 42 iterations as the RL and NRL models did, this training would have taken over a month and half.

Chapter 5

Conclusion

This thesis focused on the comparison between econometric dynamic discrete choice models, and IRL maximum entropy models, with specific attention given to their applications to the route choice problem. Furthermore, we provided descriptions of the algorithms and models using common notation to facilitate mutual understanding between the two fields. By creating this understanding, we hope that the two approaches may be able to take and build upon those from across this bridge, leading to wider corpora of literature available from both the past and the future.

This thesis served to achieve two goals. Firstly, we provided an introduction for researchers from one field to build a framework of knowledge to understand the approaches used by the other field. While there have been surveys and tutorials written for RCMs and the variety of approaches (Zimmermann and Frejinger, 2020; Prato, 2009), the emphasis we put towards describing the two fields of econometrics and IRL as well as the depth with which we describe how these fields approach the route choice problem are made to specifically cater towards researchers from one area attempting to understand the other. By utilizing a common notation, researchers should be able to familiarize themselves quickly with approaches novel to them, as they can compare the models and algorithms to those they already have knowledge of. Secondly, we provided a comparison between the two fields, through our reported results and use of various metrics taken from each of the fields. With use of the shared metrics, we provided values from our testing that can contextualize the results reported in previous work.

Of the issues encountered in this work, the primary barrier to the achievement of all objectives set out in this thesis was the issues surrounding our numerical results. Throughout the implementation process, our algorithms were plagued with numerical instabilities, our approaches to fixing were described in detail in the previous section. With our transformation of the algorithms to the log space, we found greater stability, but still had convergence issues with the Portland dataset.

There are many different directions for future work based on the initial information presented in this thesis. Completing the computations and testing on the datasets would be a clear starting point, but a further investigation into the exact similarities between the Nested Recursive Logit and the Generalized Maximum Causal Entropy models may have merit. Additionally, creating more comparisons between models from separate fields tackling the route choice problem would help expand on the primary objective set out by this thesis, bringing a wider scope of literature for researchers to incorporate into their own future work.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 1, New York, NY, USA, 2004. ACM.

Bekhor, S., Ben-Akiva, M., and Ramming, M. Evaluation of choice set generation algorithms for route choice models. *Annals OR*, 144:235–247, 2006.

Bellman, R. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503 – 515, 1954.

Bellman, R. On A Routing Problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

Ben-Akiva, M. and Bierlaire, M. Discrete Choice Methods And Their Applications To Short Term Travel Decisions. *Handbook of Transportation Science*, 26, 2000.

Bertsekas, D. P. and Tsitsiklis, J. N. An Analysis of Stochastic Shortest Path Problems. *Math. Oper. Res.*, 16:580–595, 1991.

Burton, D. and Toint, P. On an instance of the inverse shortest paths problem. *Mathematical Programming*, 53:45–61, 1992.

Cascetta, E., Nuzzolo, A., Russo, F., and Vitetta, A. A modified logit route choice model overcoming path overlapping problems. Specification and some calibration results for interurban networks. In *Transportation and Traffic Theory. Proceedings of The 13th International Symposium On Transportation And Traffic Theory, Lyon, France, 24-26 July 1996*, 1996.

Chintagunta, P. and Nair, H. Discrete-Choice Models of Consumer Demand in Marketing. *Marketing Science*, 30:977–996, 2011.

Cooper, A. and Millspaugh, J. The Application of Discrete Choice Models to Wildlife Resource Selection Studies. *Aspen Bibliography*, 80, 1999.

Fosgerau, M., Frejinger, E., and Karlstrom, A. A link based network route choice model with unrestricted choice set. *Transportation Research Part B: Methodological*, 56:70 – 80, 2013.

Mai, T., Fosgerau, M., and Frejinger, E. A nested recursive logit model for route choice analysis. *Transportation Research Part B: Methodological*, 75:100–112, 2015.

Mai, T., Bastin, F., and Frejinger, E. A decomposition method for estimating recursive logit based route choice models. *EURO Journal on Transportation and Logistics*, 7(3):253–275, 2018.

Mai, T., Chan, K. Y., and Jaillet, P. Generalized Maximum Causal Entropy for Inverse Reinforcement Learning. *ArXiv*, abs/1911.06928, 2019.

McFadden, D. Modelling the Choice of Residential Location. Cowles Foundation Discussion Papers 477, Cowles Foundation for Research in Economics, Yale University, 1977.

Nocedal, J. and Wright, S. *Numerical optimization*. Springer New York, 2006.

Prato, C. G. Route choice modeling: past, present and future research directions. *Journal of Choice Modelling*, 2(1):65 – 100, 2009.

Russell, S. Learning Agents for Uncertain Environments (Extended Abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, page 101–103, New York, NY, USA, 1998.

Rust, J. Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher. *Econometrica*, 55(5):999–1033, 1987.

Rust, J. Chapter 51 Structural estimation of markov decision processes. volume 4 of *Handbook of Econometrics*, 3081–3143. Elsevier, 1994.

Shannon, C. E. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

Train, K. E. *Discrete choice methods with simulation*. Cambridge university press, 2009.

Vovsha, P. Application of Cross-Nested Logit Model to Mode Choice in Tel Aviv, Israel, Metropolitan Area. *Transportation Research Record*, 1607(1):6–15, 1997.

Wasi, N. and Keane, M. P. Estimation of Discrete Choice Models with Many Alternatives Using Random Subsets of the Full Choice Set: With an Application to Demand for Frozen Pizza. Economics Papers 2012-W13, Economics Group, Nuffield College, University of Oxford, 2012.

Ziebart, B., Bagnell, J., and Dey, A. Modeling Interaction via the Principle of Maximum Causal Entropy. In *ICML*, 1255–1262, 2010b.

Ziebart, B. D. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, 2010.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. Maximum Entropy Inverse Reinforcement Learning. In *Proc. AAAI*, 1433–1438, 2008.

Zimmermann, M. and Frejinger, E. A tutorial on recursive models for analyzing and predicting path choice behavior. *EURO Journal on Transportation and Logistics*, 9(2):1–12, 2020.