A National Disaggregate Transportation Demand Model for the Analysis of Autonomous Taxi Systems

Alexander Penn Hill Wyrough, Jr.
Advisor: Professor Alain L. Kornhauser *71

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering
Department of Operations Research and Financial Engineering
Princeton University

June 2014
I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

__________________________

Alexander Penn Hill Wyrough, Jr.

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

__________________________

Alexander Penn Hill Wyrough, Jr.
Abstract

Around the world every day, nearly one billion personal automobiles and other motorized vehicles travel millions of miles of roadway. The world’s drivers spend trillions of dollars on these vehicles, which generally utilize automotive systems and technologies that were developed more than 50 years ago. The additional indirect costs of traffic congestion, pollution, and accidents add many tens of billions of dollars in expenses to the societal bill for personal transportation freedom. As with many other industries, new technologies have emerged with the potential to revolutionize old models and deliver greater individual and collective good at lower cost. These novel technologies can facilitate the creation of fleets of autonomously driven cars to serve individual transportation demand and replace the personal automobile. One of the more vexing challenges of the adoption of self-driving cars is not the technological solution, but the determination of the likely personal demand and expected patterns of use required to design the optimal self-driving automotive system.

To that end, modeling existing personal travel behavior is fundamental to the creation of self-driving automotive systems. This thesis develops an existing disaggregate travel demand model, constrained for use within the state of New Jersey, into a daily transportation demand model for the entire United States. Data drawn from the U.S. 2010 Census, as well as other sources, are used to simulate 308,745,538 synthetic individuals with specific personal attributes and create the 1,009,322,835 automotive trips these individuals take across the U.S. on a typical work day. With precise spatial and temporal attributes that mimic actual personal travel behavior, these trips comprise a ready data set to analyze the efficacy of novel transportation systems. After determining where each individual wants to go, from where, and when, one can begin to engineer systems to serve this demand using autonomously driven vehicles. Specifically, this data set is designed to gain insight into a national system of autonomous taxis that could (a) match the comfort and convenience of personal cars, (b) exceed the accessibility of mass transit, and (c) deliver wide-ranging benefits such as alleviated congestion, reduced pollution, and increased vehicle safety.
Acknowledgements

First, I would like to thank my advisor, Professor Alain Kornhauser. As a teacher and trusted advisor for my years navigating Princeton, Prof. K has been the ultimate example of intellectual energy and curiosity to which I aspire. His knowledge and indomitable passion have been the source of guidance and inspiration that made this work possible. I also thank Talal Mufti, Jake Gao, and the history of ORF467 classes who have unknowingly paved the way for my thesis.

I would also like to thank my family. Mom and Dad: Thank you for your unending well of support, total love, and devoted generosity. I am so very grateful for the opportunities you have given to me and enabled me to pursue. Sarah and Stu: You two are just the bomb. I love you all dearly. I also must thank my grandparents, Nonna and Pappo, for always being there throughout my time at Princeton. You have always made me feel close to home.

Finally, I would like to thank all of my friends - in the Tig, in the Torch, and everywhere in between - who have been the greatest hallmark of my collegiate experience and collectively offered me such valuable, devoted, continued, and enjoyable companionship through it all.
Contents

<table>
<thead>
<tr>
<th>Abstract</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>iv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
</tbody>
</table>

1 Introduction 1
1.1 Motivation 3
1.2 Travel Demand Modeling Methodologies 5
  1.2.1 Introduction to Activity-Based Demand Modeling 6
  1.2.2 Agent-Based Models and Home-Based Models 7
  1.2.3 The Workday as a Unit of Analysis 7
  1.2.4 Disaggregate Demand Modeling in ORFE 8
1.3 A National Model 8
  1.3.1 Review of New Jersey State Model 8
  1.3.2 National Model Objectives 9

2 Project Methodology 10
2.1 Task 1: Generation of Populace 10
2.2 Task 2: Workplace Assignment 14
2.3 Task 3: School Assignment 17
2.4 Task 4: Tour Assignment and Activity Patterns 19
2.5 Task 5: Trip Destination Assignment 20
  2.5.1 Mode Split for Long-Distance Trips: Air Travel 21
2.6 Task 6: Arrival and Departure Time Assignment 23
2.7 Geographical Assumptions of the Model 25

3 Data 28
3.1 Task 1 28
  3.1.1 2010 United States Census Summary File 1 Data 28
  3.1.2 2010 American Community Survey of Household Income 29
3.2 Task 2 30
  3.2.1 2010 American Community Survey Journey-to-Work Census 30
  3.2.2 2010 American Community Survey Industry Type Participation by Gender and Median Income 30
  3.2.3 Employee and Patronage Data 31
3.3 Task 3 33
  3.3.1 National Center for Education Statistics 33
  3.3.2 Data for Post-Secondary Schools 33
3.4 Task 4 34
3.5 Task 5 34
3.6 Task 6 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Synthesizer Analysis and Results</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Demographic Characteristics of Simulated Data</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Home to Work Trips</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Home to School Trips</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>All Daily Trips</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Time Distribution of Trips</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>Case Studies of Simulated Trips</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>Manhattan, New York</td>
<td>61</td>
</tr>
<tr>
<td>5.2</td>
<td>Fargo, North Dakota</td>
<td>63</td>
</tr>
<tr>
<td>5.3</td>
<td>Peoria, Illinois</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>Limitations, Next Steps, and Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Limitations</td>
<td>67</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Data Scope and Applicability</td>
<td>67</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Run Time and Memory Limitations</td>
<td>69</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Independence Among States</td>
<td>71</td>
</tr>
<tr>
<td>6.2</td>
<td>Next Steps</td>
<td>71</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Network Assignment</td>
<td>72</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Multi-Modal Transportation</td>
<td>72</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Non-Resident Travel Demand</td>
<td>73</td>
</tr>
<tr>
<td>6.3</td>
<td>Conclusion</td>
<td>73</td>
</tr>
<tr>
<td>A</td>
<td>Technical Documentation of the Synthesizer</td>
<td>74</td>
</tr>
<tr>
<td>A.1</td>
<td>Sample Execution</td>
<td>74</td>
</tr>
<tr>
<td>B</td>
<td>Source Code</td>
<td>76</td>
</tr>
<tr>
<td>B.1</td>
<td>Module 1 Source Code</td>
<td>76</td>
</tr>
<tr>
<td>B.2</td>
<td>Module 2 Source Code</td>
<td>88</td>
</tr>
<tr>
<td>B.2.1</td>
<td>Module 2 Industry Selection</td>
<td>91</td>
</tr>
<tr>
<td>B.2.2</td>
<td>Module 2 Employer Selection</td>
<td>94</td>
</tr>
<tr>
<td>B.3</td>
<td>Module 3 Source Code</td>
<td>96</td>
</tr>
<tr>
<td>B.3.1</td>
<td>Module 3 School Selection</td>
<td>100</td>
</tr>
<tr>
<td>B.4</td>
<td>Module 4 Source Code</td>
<td>108</td>
</tr>
<tr>
<td>B.5</td>
<td>Module 5 Source Code</td>
<td>110</td>
</tr>
<tr>
<td>B.5.1</td>
<td>Mode Split for Air Travel</td>
<td>113</td>
</tr>
<tr>
<td>B.6</td>
<td>Module 6 Source Code</td>
<td>117</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Traveler Types, Household Types, and Income Brackets .................. 12
2.2 Daily Trip Tours Definitions ........................................... 20

3.1 Summary of Module 1 State Data Input Files .......................... 29
3.2 NAICS Industries Used .................................................. 32
3.3 Trip Tour Distributions ................................................ 35
3.4 Bell Times and Durations by Industry ................................. 36

4.1 Summary of South Carolina Generated Population ....................... 39
4.2 Summary of Commuting Trips by State ................................ 44
4.3 Addison County Worker Flows ......................................... 48
4.4 Summary of School Trips by State ..................................... 53
4.5 Mercer County Public High School Enrollment Analysis ............. 54
4.6 Total Number of Trips Taken by State Residents ....................... 57

5.1 oTrip Statistics for Manhattan, NY .................................... 61
5.2 oTrip Statistics for Fargo, ND ......................................... 64
5.3 oTrip Statistics for Peoria, IL ......................................... 65
List of Figures

2.1 Task 1 Sample Output for Oregon Residents ........................................... 13
2.2 The Gravity Model For Assigning County of Work ................................. 15
2.3 The Gravity Model For Assigning Industry of Work ............................... 16
2.4 The Gravity Model For Assigning Place of Work ................................ 16
2.5 Task 2 Sample Output for West Virginia Residents ................................ 17
2.6 The Gravity Model For Assigning Schools ............................................. 18
2.7 Task 3 Sample Output for Tennessee Residents ......................................... 18
2.8 Sample Output for Indiana Resident ....................................................... 21
2.9 Sample Output for Air Travel Mode Split .............................................. 22
2.10 Sample Output for Mississippi Resident ................................................ 24
2.11 Relative Geography of Block Groups (Blue) Within Census Tracts (Black) Within a County (Red) ................................................................. 26
2.12 Relative Geography of Census Tracts Within Counties ............................. 27
2.13 Relative Geography of Counties Within Continental U.S. ......................... 27
3.1 Empirical Distribution of Daily Trips by Start Time ................................ 35
4.1 Comparison of Male Populations in Simulated Data and 2010 Census Data ...... 38
4.2 Comparison of Female Populations in Simulated Data and 2010 Census Data .... 38
4.3 Cumulative Distribution of South Carolina Household Income Simulated and Actual Data .......................................................... 39
4.4 Distribution of Simulated South Carolina Household Incomes .................... 40
4.5 Cumulative Distribution of Trip Length For Commutes by State (AL - CO) .... 41
4.6 Cumulative Distribution of Trip Length For Commutes by State (CT - HI) .... 41
4.7 Cumulative Distribution of Trip Length For Commutes by State (ID - KY) .... 41
4.8 Cumulative Distribution of Trip Length For Commutes by State (LA - MN) .... 42
4.9 Cumulative Distribution of Trip Length For Commutes by State (MS - NH) .... 42
4.10 Cumulative Distribution of Trip Length For Commutes by State (NJ- OH) .... 42
4.11 Cumulative Distribution of Trip Length For Commutes by State (OK - SD) .... 43
4.12 Cumulative Distribution of Trip Length For Commutes by State (TN - WA) .... 43
4.13 Cumulative Distribution of Trip Length For Commutes by State (WV - WY) .... 43
4.14 National Average Commute Travel Time .............................................. 45
4.15 Median Trip Length by County in New York State .................................. 46
4.16 Worker Flows in Addison County (VT) .................................................. 47
4.17 Sample Commuting Trip Filaments for Addison County (VT) .................... 49
4.18 Cumulative Distribution of Trip Length For School Trips by State (AL - CO) .. 50
4.19 Cumulative Distribution of Trip Length For School Trips by State (CT - HI) .. 50
4.20 Cumulative Distribution of Trip Length For School Trips by State (ID - KY) .. 50
4.21 Cumulative Distribution of Trip Length For School Trips by State (LA - MN) .. 51
4.22 Cumulative Distribution of Trip Length For School Trips by State (MS - NH) .. 51
4.23 Cumulative Distribution of Trip Length For School Trips by State (NJ- OH) .. 51
4.24 Cumulative Distribution of Trip Length For School Trips by State (OK - SD) .. 52
4.25 Cumulative Distribution of Trip Length For School Trips by State (TN - WA) .. 52
Chapter 1

Introduction

Most daily routines involve getting from one place to another, and deciding how to move among available modes of transportation and their networks. In addition to availability constraints, the essential factors in personal transportation decision-making are time and cost. Transportation networks are subject to myriad influences, such as the availability of natural resources, the health of the economy, and technological innovation. Changes in these elements have important implications for the management of transportation systems. Moreover, the volatility of these factors challenges administrators and planners to constantly revise transportation systems in order to adapt to the ever-changing environment and deliver greater personal mobility. Essential to this responsibility is the study of the feasibility of innovative modes of transportation. Modeling the current travel behavior of the population is the crucial first step in this regard. This thesis creates an activity-based model and methodology to simulate the individual daily travel behavior of all U.S. residents for the purpose of examining the feasibility of novel modes of personal rapid transportation on a national scale.

This thesis concerns itself with disaggregate personal trip patterns, which is to say the trips taken by, and motivating factors for, a specific individual, rather than broad travel trends for the U.S. population. This focus is motivated by a singular objective: the full potential of an innovative transportation system relies on the manner and extent to which it can service the daily transportation needs and considerations of individuals. A macroscopic model for transportation demand, which often lacks the specificity of temporal and spatial attributes of trips, is not be appropriate for such an aim.

Modeling an individual’s daily trips requires modeling the travel demand for individuals. This
thesis uses demographic, business activity, and school enrollment data to create and match disaggregated travel supply (U.S. residents) to travel demand, where the sum of the disaggregate travel behavior matches that of the United States at large. In 2009, the average U.S. resident took three to four trips throughout the day and traveled 36 miles. In 2010, U.S. residents and foreign visitors traveled 4.6 trillion passenger miles; close to 80% of this movement is due to personal, non-commercial trips. (U.S. Department of Transportation, 2013) It is these personal trips - the daily excursions to work, school, or the supermarket - that are the subject of this thesis’ simulation. Routine activity-based travel is deeply embedded in our lives; this predictability presents an opportunity to model behavior and understand how people move today.

The primary mode of transportation is the personally-owned automobile. Statistics from the 2009 National Household Travel Survey demonstrate the absolute dominance of the automobile in daily commutes and also highlight the huge opportunity for alternative modes of personal transportation. (U.S. Department of Transportation, 2009) As a percentage of worker commutes, personal vehicles have been the mode of choice for nearly 90% of trips in the past 40 years. The next most popular option, mass transit, represents less than 5% of those trips. The automobile has triumphed as a mode of transportation because it delivers passengers to precise locations on demand. However, automobile networks and traditional road travel, while substantially more convenient than most alternatives, are far from ideal. Mashayekh et al. describe the particular deficiencies of automobile dependent transportation systems: “There are continuing concerns for secondary effects including accidents, air emissions, congestion, lack of physical exercise, mobility for those without motor vehicles, noise, petroleum dependence, and urban sprawl.” (Mashayekh et al., 2011) The underlying objective of this thesis is to advance alternative solutions that can reasonably address the demand currently serviced by traditional automobiles, and alleviate these external costs.

In summary, the purpose of this thesis is to develop a framework for large-scale personal transportation demand modeling within the United States through a disaggregate, activity-based demand model and contribute to the necessary analysis for the implementation of game-changing self-driving transportation systems. Without the ability to accurately model current transportation behavior on existing networks, no proper investigation into the application and responsiveness of original solutions to meet the mobility demands of a population can be accomplished. Understanding the need for such systems is the first motivation for this work.
1.1 Motivation

Currently, novel transportation development has reached a particular nexus. The technology for modes of autonomous personal transit is demonstrably feasible in the near term. Rising costs of fuel, persistent levels of pollution, and unyielding accident rates put continued pressure on policy makers and automobile manufacturers to improve modes of personal transportation. The opportunity to leverage emerging transportation technologies such as self-driving, automated vehicles has presented itself. Nissan, one of the largest foreign automobile manufacturers, has followed General Motors' lead and recently announced its intention to release models of autonomously-driven vehicles by 2020. (Shankland, 2013) The Google Car Project has guaranteed a commercial self-driving product by 2017; its completely autonomous vehicles prototypes have logged in excess of 300,000 miles of driving as of 2012. (Murray, 2012) Self-driving cars are clearly no longer just a dream.

If nothing else, the Google Car Project shows self-driving cars can make the roads much safer for everyone. Autonomous vehicle driving, by its nature, removes human error, which is responsible for over 90% of auto accidents and the congestion caused by unnecessary breaking and uneven speeds. (Treat et al., 1979) Sebastian Thrun, the Google Car Project lead developer, claims that the use of its vehicle can reduce the number of cars on the road by 90% while reducing traffic accidents, fuel consumption, and wasted commute time by similar amounts. (Mui, 2013) Self-driving car chronicler, Chunka Mui, translates these projections into yearly metrics: 30,000 lives saved, $400 billion saved in accident related costs, and a substantial reduction in the 4.8 billion hours and 1.9 billion gallons of fuel wasted in traffic congestion. (Mui, 2013) In 2014, the Federal Highway Administration plans to spend $41 billion for projects to improve highway safety in a losing battle, as automobile accident rates have been relatively stable. (U.S. Department of Transportation, 2014) The potential to reduce the widespread costs of the traditional automobile system while delivering personal mobility freedom to consumers is a motivation for the work done by Google, automobile manufacturers, policy makers and this thesis.

It is not difficult to see why the traditional personal ownership model of automobiles is inefficient. Personal transportation (i.e. automobile ownership and use) ranks as the second largest household expenditure yearly, and yet the car sits unused in the home or parking lot some 90% of the time. (Burns et al., 2013) Moreover, as of 2012, over 75% of workers commute alone; the percentage of commuters traveling to work alone has been steadily rising since the 1960’s. (Shah, 2013) In total, nearly 40% of all car trips made in the U.S. are made with just one person in the vehicle, according to the RITA Bureau of Transportation Statistics. The huge personal and societal costs
of such automobile use has prompted the widespread observations that, ”roadway transportation is now as ripe for transformation [sic] as the telecommunications, photography, computer, media, television and pharmaceutical industries were over the past two decades.” (Burns et al., 2013) The development of self-driving car technologies is catalyst for a revolution in personal transportation.

The inefficiencies in the use of personal automobiles have stimulated the conceptual development of autonomous taxis or aTaxis. An aTaxi is a self-driving vehicle that services customers by taking individuals from their personal points of origin to their destinations. An aTaxi system is a ubiquitous network of aTaxi stands, where aTaxis pick up passengers at one stand and deliver them to the stand closest to their destination, with minimal additional walking needed at either end of the network. Specifically, the aTaxi system considered in this thesis is a system of aTaxi stands intended to serve the entire personal transportation demand of the United States. To create a national aTaxi system, U.S. would be partitioned into a grid of square blocks, each 0.25 square miles in size, with an aTaxi stand located at the center of each block. With this system, a passenger can travel from origin to destination in an aTaxi, and not have to walk more than one half mile. This system would allow for easy ride-sharing: passengers traveling between common origins and destinations at the same time would share an aTaxi.

The proposed system of autonomous taxi stands would leverage autonomous vehicles and ride-sharing potential for numerous benefits. Passengers are given the traditional comfort of automobiles with flexible and dynamic service. Autonomous vehicles free passengers from having to devote all of their attention to the task of driving. Time otherwise spent driving could now be used much more efficiently. In addition, aTaxi’s improve the overall efficiency of the macroscopic transportation network when compared to current transit systems. Self-driving cars are safer and can use platooning techniques to reduce overall congestion. Plus, every aTaxi ride-sharing opportunity is an opportunity to take a car off the road. Increasing average vehicle occupancy (AVO) has many positive effects, namely reduced congestion, lower environmental impact, and decreased cost of fuel per person. The proposed aTaxi system matches the comfort and convenience of personal automobile ownership, without the considerable costs of personal ownership, and can exceed the efficiency, speed, and accessibility of current mass transit.

However, no system as described here could possibly be implemented without first understanding where people want to go - from where, to where - and when they want to go. A model that does not describe trips with specific spatial and temporal detail is useless in analyzing a system intended to service individual trips and compete with traditional automobiles. This thesis is the second generation in a series of efforts made to examine the feasibility of autonomous taxi systems within a
large geographical region. Talal Mufti, who performed the task originally for the state of New Jersey, described his motivation: “Determining where to place access points [to alternative personal rapid transit systems] to meet demand is pivotal in competing with the automobile, and doing so requires information and a level of detail that no survey can provide.” (Mufti, 2012, p. 7). This thesis aims to create a model with an adequate level of realistic detail to inform next generation transportation systems for the United States as a whole.

1.2 Travel Demand Modeling Methodologies

Travel demand modeling is an important instrument for making informed decisions about transportation system infrastructure and policies. The goal of the models is to predict travel behavior under certain conditions. Broadly, travel demand modeling is used to “analyze the response of users to changes brought about by new services, investments in infrastructure and to changes in operating and pricing policies.” (Ben-Akiva and Lerman, 2000, 1.1) Forecasting models were first employed to analyze capital investments in national highway systems and their relationship with land-use policies. As processing power has increased and cheaper memory has become available, modeling has moved away from the long-term policy motivations towards analyzing more complex, multi-modal urban transportation systems.

Early transportation forecasting, using a Trip-Based approach, centered on a traditional four-step model. The foundation of this model is the segregation of the geographical region in question into designated areas of homogeneous land-use purposes. For example, a county would be partitioned into Traffic Analysis Zones (TAZ), characterized by their use - residential, commercial, recreational, etc. The four-step model to forecasting trips taken between each TAZ involves estimating the number of trips generated by each TAZ, the distribution of trips taken between each TAZ, and then further specifying those trips by mode and by route taken.

This Trip-Based model has been widely used for macroscopic analysis of transportation networks. It is a simple and aggregate approach to transportation that has offered valuable opportunities for analysis of group travel behavior, and the implication of land-zoning policies. However, there are significant shortcomings to this model, chiefly:

A fundamental conceptual problem with the trip-based approach is the use of trips as the unit of analysis. Separate models are developed for home-based trips and non-home based trips, without consideration of dependence among such trips. Further, the organization (scheduling) of trips is not considered; that is, there is no distinction between home-based trips made as part of a single-stop sojourn from home and those made as part of a multiple-stop sojourn from home. (Bhat and Koppelman, 1999)
In short, the traditional model does not illuminate true personal travel behaviors. Trip-Based models fail to follow an individual through his or her daily trip tour, and so there is no consideration for the interdependence of an individual’s trips with respect to time and space. Aggregate forecasting models are insufficient for the analysis of an aTaxi system, which is entirely dependent on the exact temporal and spatial distribution of trips. A more realistic approach would consider the needs of the travelers to determine their trips and how the trips are organized in time and space.

### 1.2.1 Introduction to Activity-Based Demand Modeling

Activity-Based Demand Modeling is the application of discrete choice analysis methods to model the decision making process that motivates daily trip tours. (Ben-Akiva and Lerman, 2000) A trip tour is the sequence or chain of trips in time and space throughout a particular day. While these methods are fundamentally different than Trip-Based Models, the objective is identical when viewed holistically. Both seek to ensure that the set of the individual units of analysis resembles true aggregate behavior.

Central to the notion of an Activity-Based Demand Model is the idea of transportation, or travel, as a derived demand. To wit, travel is not an end in and of itself, rather it is a by-product of the need to be at work, or attend school, or run an errand and pursue all of our other activities. As Bhat and Koppelman put it: “The conceptual appeal of this approach originates from the realization that the need and desire to participate in activities is more basic than the travel that some of these participations may entail.” (Bhat and Koppelman, 1999) Whereas earlier models used land-use as the central motivating factor, activity based models use personal need. At their core, these models presuppose a predictable nature of daily trip patterns, particularly those anchored in time and space, such as work and school. Given the goal of analyzing the participation in a nationwide aTaxi system, placing the emphasis on the individual, and his or her activity choices, sensible and appropriate.

There are a variety of Activity-Based Demand Models that derive travel demand using a host of considerations. One of the more successful models has been to use a schedule modeling system, with a programmatic approach to daily time planning and patterns. The model, called a Comprehensive Econometric Microsimulator for Activity-Travel Patterns (CEMDAP), builds a trip tour where the attributes “characterize a continuous time activity-travel pattern built within the space-time constraints imposed by work and school.” (Bhat and Pinjari, 2011) In other modeling techniques, schedules are solved with linear programming to order tasks within a day, just as one would order manufacturing tasks. While no technique used is exhaustive in its application of the available data,
agent-based models, which imply a certain, yet broad, framework, permit the most comprehensive simulations.

1.2.2 Agent-Based Models and Home-Based Models

Agent-Based Models “incorporate the complexity of human behavior using ‘agents’ that are autonomous and interactive [sic] in nature” in order to simulate their decisions over time. (Bhat and Pinjari, 2011) This class of models integrates activity and time-use scheduling techniques from other demand models, yet remains focused on the individual as the unit of analysis, rather than other subsets of the population. To no surprise, the primary fault of these models is the inability to adequately incorporate interactions between agents, such as those who live in the same house or tenant building, or work in the same office. However, Agent-Based Models represent the most promising foundation towards comprehensively modeling activity-based demand, especially when used in conjunction with home-centric or home-based models. In response, this thesis uses a home-based model - where all trip tours begin and end at home - where residents are grouped by household to maintain the opportunity for including household dynamics and trip correlations.

In current research, most models, if not all, are narrowly applied to urban areas, cities, or at largest, counties. A model does not currently exist that can be applied to large-scale populations or generic areas. The goal of the demand model created by this thesis is to combine the best of agent-based models, while using data and techniques to ensure broad applicability. This premise is the only one suitable for the building of a large-scale disaggregate model to uniformly simulate individual trip tours within the U.S.

1.2.3 The Workday as a Unit of Analysis

Many travel demand models, including this one, use the typical weekday as the basic time frame of analysis. There is no doubt that significant variability exists across weekdays for many people. An examination of a reader’s own travel behaviors would reveal as much. In addition, weekend travel is not an inconsequential component of overall travel behavior. However, the predictability of weekday trips, and the anchors of those trips (school, work), require the “implicit assumption [that] there is little variation in activity-travel patterns across different days of the week.” (Bhat and Koppelman, 1999) As a foundation, the typical workday simulation is a prototype of activity-based travel. From there, models can begin to introduce complexities such as multi-day trip tours or the Friday afternoon rush.
1.2.4 Disaggregate Demand Modeling in ORFE

The project undertaken by this thesis is the second evolution in a series of efforts to examine the feasibility of autonomous taxi systems. Within the context of Princeton University’s ORF467: Transportation Systems Planning and Analysis class, offered by the Department of Operations Research and Financial Engineering, the original work of this project began in 2011. The goal was to generate a simulation of all the trips taken by the 8.9 million residents and workers in New Jersey on a typical workday. This project became the work of Princeton University Master’s student, Talal Mufti, for his thesis. His groundwork and methodology are the forebears of this thesis. Mufti’s original data set has undergone improvements in subsequent years, particularly through the work of Jingkang Gao for his senior thesis in 2013. In the Fall of 2013, the ORF467 class continued to add complexities to the model, such as multi-modal travel. An analysis of the data set demonstrated great potential for both the modeling methods and the proposed aTaxi system.

Importantly, the New Jersey State Model model produced results of significant relevance to travel within New Jersey, particularly with respect to the New Jersey Transit rail network. The behavior and patterns of the simulated population have the potential to inform policy decisions on investment in the New Jersey rail system. Lastly, thought naturally extended towards the application of the model to all states within the U.S. to comprise a national model of travel demand. This thesis initiates the effort to build a personal trip data set for the entire county and extends the New Jersey State Model, the data used, and its programming foundation.

1.3 A National Model

1.3.1 Review of New Jersey State Model

In order to situate this thesis squarely in its place in the evolution of disaggregate transportation demand modeling within ORFE, the New Jersey State Model is briefly revisited. The New Jersey State Model builds a population using highly localized data, making the assumption that the attributes of residents - age, gender, income, location - can explain their daily travel demand. The model then satisfies this demand by ascribing all the trips with specific locations of employment, school, and patronage and assigning times.

Because of the narrow focus of the New Jersey State Model, simplifying assumptions were made regarding out-state-travel. The model created eight nodes of out-of-state travel - including New York City and Philadelphia - that were the synthetic points of origin and destination for all travel
outside of New Jersey. For instance, all travel in and out of New Jersey to neighboring Bucks County in Pennsylvania went to the centroid of the county, rather than to individual homes or workplaces.

Gao’s work made several significant improvements to the raw data of the New Jersey State Model, modifying employee and patronage ratios as well as school enrollment, to better match New Jersey’s population. In addition, he improved the temporal distribution of trips to more closely correspond to true travel behaviors. His improved data and techniques were applied to Mufti’s original trip synthesizer to produce an even more realistic data set.

1.3.2 National Model Objectives

The objective of this thesis is to extend the New Jersey state trip synthesizer model developed originally by Talal Mufti and the 2011 ORF467 class, and enhanced by Jingkang Gao in 2013, to apply to any state within the U.S. This thesis uses the same methods in spirit of the New Jersey State Model and begins with that framework. The specific objectives to augment the past model can be outlined in three ways:

- **Data Objectives**: A national model requires data on a much larger, and comprehensive scale. The data used for New Jersey are extremely small in comparison, were adjusted manually, and have no uniformity.
- **Modeling Objectives**: The New Jersey State Model is hard-coded for the use in New Jersey. Its one state focus simplifies interstate travel. A national model requires generalization and new considerations, such as international travel.
- **Programming Objectives**: The software component to the synthesizer is hard-coded for New Jersey state and has significant simplifying assumptions. Software applicable to each state requires wide adaptation and more robust methods to handle new data inputs and the increased scope of the model.

The task of extending the trip synthesizer nationally first requires a much broader initial data set. Data sets for all 50 states and D.C. are prepared. In addition, when New Jersey was the area of focus, inter-county travel was primary, and interstate travel was simplified. In the national scheme, travel between counties and states is treated identically. However, despite the overhauls, this project is an extension of previous work and so maintains the task and module schematic of the New Jersey State Model.
Chapter 2

Project Methodology

This chapter details the framework, assumptions, and procedural method for creating a synthesized set of personal trips taken by each person living in the United States. The methodology is characterized by six separate tasks, each achieving a distinct goal and each requiring unique data sets. In Chapter 3, explanation of all relevant data sources and inputs parallel this task structure.

The methodology is a process of iterative construction that begins with demographic data drawn from the U.S. Census Bureau, which details hyper-local personal and housing data. First, a simulated population of residents within the U.S. is created. With attributes as general as sex and age, together with personal income and residence within a particular household, the synthetic population is built to mimic the U.S. population as it was in 2010. With the travel supply created, workforce participation, school enrollment, and patronage data provide the distribution of demand. Through matching of travel demand and travel supply under a set of assumptions about travel behavior, the daily trip tour of each individual is generated. The final step of disaggregation gives the trips a temporal component. Collectively, the steps create a comprehensive data set of all of the casual, non-commercial, trips, characterized by time and space, taken within the United States on an average workday day.

2.1 Task 1: Generation of Populace

The first task is the generation of the approximately 300 million residents of the United States. This fundamental first step creates participants within the contrived system and gives them basic, personal attributes that will inform their daily activity patterns. The challenge in this step is recreating a massive population with enough specificity to characterize individuals in a truly disaggregate manner. The model must balance the requirements of an Agent-Based model and the availability of
data that will allow for both breadth and depth in simulation. For this reason, the concept of the Census block is an essential cornerstone of Task 1. As defined by the U.S. Census Bureau, a Census block is “the smallest geographic unit used... for tabulation of 100-percent data (data collected from all houses, rather than a sample of houses).” (SF 1 Technical Documentation, 2012) Task 1 recreates the population from the ground up, iterating through each of the 11,078,297 Census blocks within the U.S., using demographic and housing data available at the block level.¹

Census blocks are normally bound by streets, and populated with fewer than 100 people. The first level of aggregation in this model is the assumption that every resident within a Census block lives at the geographical centroid of that Census block and his or her characteristics follow the distributions implied by the Census data. There is no widely available data source that would permit further disaggregation beyond what normally amounts to an acre in area. For a full summary of the relative geographies of, and assumptions related to, Census areas used within this method refer to Section 2.7.

Task 1 assigns traveler and household designations to the simulated population based on their attributes and characterizations of the national workforce and student populations. These manufactured rubrics, shown in Table 2.1, are used to filter personal attributes into expected trip patterns in later steps. The Traveler Type categories were first created by Mufti and are re-used with one modification: the national model removes the Traveler Type 7, an out-of-state traveler, because it is no longer relevant. The percentages used in the Traveler Type Classification are drawn from workforce data and involve various assumptions about unemployment and sick-days to correct for the right number of workers traveling to work on an average day.² The Household Type categories, as well as the Income Bracket categories, are the same ones used within the United States Census.

Task 1 follows a step-by-step procedure for each Census block. Task 1 first assigns residents an age and gender simultaneously, ensuring that total populations by age bracket and gender match the 2010 population exactly. This is done by drawing from the distribution of residents in each age bracket by gender with replacement. The U.S. Census designates age brackets that normally span five years and groups each resident within an age bracket. To overcome this slight aggregation, age is uniformly sampled at random within each age group.

Secondly, Task 1 populates a set of residents who live within group quarters. Group quarters is the designation of special living arrangements that fall under Household Types two through eight, detailed in Table 2.1. Each Census block has anywhere from zero to all of its residents living within group quarters. Task 1 fills the group quarters quota for each Census block by drawing from the

---

¹See Table 3.1.1 in Section 3.1
²For an exact explanation of the Traveler Type distributions, refer to Mufti’s Master’s Thesis
Table 2.1: Traveler Types, Household Types, and Income Brackets

<table>
<thead>
<tr>
<th>Traveler Type Code</th>
<th>Traveler Type Name</th>
<th>Classification (Age, HHT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Does-Not-Travel</td>
<td>0-5, 79; 2,3,4,5,7</td>
</tr>
<tr>
<td>1</td>
<td>Student-Non-Worker</td>
<td>5-15, 16-18 x 99.81%</td>
</tr>
<tr>
<td>2</td>
<td>Student-Worker-In-County</td>
<td>16 - 18 x 0.198%</td>
</tr>
<tr>
<td>3</td>
<td>College-No-Commute</td>
<td>18-22 x 90.34%</td>
</tr>
<tr>
<td>4</td>
<td>College-Work-In-County</td>
<td>18-22 x 9.66%</td>
</tr>
<tr>
<td>5</td>
<td>Typical Traveler</td>
<td>22-64 x 78%</td>
</tr>
<tr>
<td>6</td>
<td>Home-Worker-Traveler</td>
<td>22-64 x 22%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HHT Code</th>
<th>Household Type Name</th>
<th>Income Code</th>
<th>Income Brackets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Family</td>
<td>0</td>
<td>&lt; $10,000</td>
</tr>
<tr>
<td>1</td>
<td>Non-Family</td>
<td>1</td>
<td>$10,000 - 14,999</td>
</tr>
<tr>
<td>2</td>
<td>Correctional Facility</td>
<td>2</td>
<td>$15,000 - 24,999</td>
</tr>
<tr>
<td>3</td>
<td>Juvenile Detention Center</td>
<td>5</td>
<td>$25,000 - 54,999</td>
</tr>
<tr>
<td>4</td>
<td>Nursing Homes</td>
<td>4</td>
<td>$55,000 - 49,999</td>
</tr>
<tr>
<td>5</td>
<td>Other Institutionalized Quarters</td>
<td>5</td>
<td>$50,000 - 74,999</td>
</tr>
<tr>
<td>6</td>
<td>Dormitories</td>
<td>6</td>
<td>$75,000 - 99,999</td>
</tr>
<tr>
<td>7</td>
<td>Military Quarters</td>
<td>7</td>
<td>$100,000 - 149,000</td>
</tr>
<tr>
<td>8</td>
<td>Other Non-Institutionalized Quarters</td>
<td>8</td>
<td>$150,000 - 199,999</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>&gt; $200,000</td>
</tr>
</tbody>
</table>

set of simulated residents, according to the U.S. Census distribution that specifies the number of residents living within each type of group quarters for three different age brackets: 17 years and younger, 18 years to 64 years old, and 65 years and older.

Thirdly, Task 1 takes the remaining population (the total population less the population in group quarters) and creates the population living within households. This step relies both on household size data - which details the number of houses in a block by their size - and the distribution of residents within family households, by relation to the householder, and non-family households. Lastly, each household is assigned an income, drawn from the American Community Survey on household income, by type (family or non-family). To assign individuals a specific income, household income is schematically distributed over the residents within a household, if the residents other than the householder logically qualify to earn an income.

Task 1 produces 51 master files containing all of the residents and their personal attributes for each state, including the District of Columbia, where residents are grouped together by block and by household. These files, and the simulated information contained within, are the foundation for the next tasks that assign each resident to their daily activities, such as work, school, and other recreational trips. An example of the output from Task 1, for Oregon (with a state code of 41), can

---

3 The total population living in a block is the sum of those in family households, non-family households, and group quarters

4 A household size is the number of people living in the house taking values of 1, 2, 3, 4, 5, 6, or 7+
be found in Figure 2.1.

<table>
<thead>
<tr>
<th>Residence State</th>
<th>County Code</th>
<th>Tract Code</th>
<th>Block Code</th>
<th>HH ID</th>
<th>HH TYPE</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Person ID Number</th>
<th>Age</th>
<th>Sex</th>
<th>Traveler Type</th>
<th>Income Bracket</th>
<th>Income Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1001</td>
<td>1</td>
<td></td>
<td>0</td>
<td>-117.562</td>
<td>41000000001</td>
<td>50</td>
<td>0</td>
<td></td>
<td>$57,557.17</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1001</td>
<td>1</td>
<td></td>
<td>0</td>
<td>-117.562</td>
<td>41000000002</td>
<td>47</td>
<td>1</td>
<td></td>
<td>$10,876.33</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1001</td>
<td>2</td>
<td></td>
<td>1</td>
<td>-117.579</td>
<td>41000000003</td>
<td>58</td>
<td>0</td>
<td></td>
<td>$119,408.49</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>3</td>
<td></td>
<td>1</td>
<td>-117.506</td>
<td>41000000004</td>
<td>92</td>
<td>1</td>
<td></td>
<td>$174,500.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>4</td>
<td></td>
<td>1</td>
<td>-117.506</td>
<td>41000000005</td>
<td>76</td>
<td>1</td>
<td></td>
<td>$265,739.90</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>5</td>
<td></td>
<td>1</td>
<td>-117.506</td>
<td>41000000006</td>
<td>55</td>
<td>1</td>
<td></td>
<td>$35,172.22</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>6</td>
<td></td>
<td>1</td>
<td>-117.506</td>
<td>41000000007</td>
<td>44</td>
<td>1</td>
<td></td>
<td>$7,800.34</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>6</td>
<td></td>
<td>1</td>
<td>-117.506</td>
<td>41000000008</td>
<td>38</td>
<td>0</td>
<td></td>
<td>$2,835.42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>7</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000009</td>
<td>84</td>
<td>0</td>
<td></td>
<td>$0.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>7</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000010</td>
<td>66</td>
<td>0</td>
<td></td>
<td>$117,427.10</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>8</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000011</td>
<td>80</td>
<td>0</td>
<td></td>
<td>$0.00</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>8</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000012</td>
<td>43</td>
<td>1</td>
<td></td>
<td>$173,315.36</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>9</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000013</td>
<td>66</td>
<td>1</td>
<td></td>
<td>$59,199.30</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>9</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000014</td>
<td>55</td>
<td>0</td>
<td></td>
<td>$3,061.48</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>10</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000015</td>
<td>78</td>
<td>0</td>
<td></td>
<td>$144,04</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>001</td>
<td>56010</td>
<td>1003</td>
<td>10</td>
<td></td>
<td>0</td>
<td>-117.506</td>
<td>41000000016</td>
<td>38</td>
<td>1</td>
<td></td>
<td>$29,752.49</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.1: Task 1 Sample Output for Oregon Residents

Task 1 relies heavily on creating, and drawing from, distributions based on Census data. This Task, and this methodology, revised the way in which these distributions were handled. For each particular Census block, all attributes associated with each person and household are drawn from the distributions in the data without replacement, so that each Census block has all of the exact characteristics of the actual block (at 2010 levels). The New Jersey State Model did so with replacement and forfeited control of the simulation. If the sampling from the actual distributions were done with replacement, the process would have rendered other distributions unusable or irrelevant. To illustrate the reasoning behind this decision, imagine a Census block where only two people lived in one household. Those two people are two females, where one is aged 40 - 44 years old, and one is aged 5 - 9 years old. If age were sampled with replacement, it would be statistically possible to obtain two females who were each less than 10 years old. This becomes troubling when building households from the simulated population within the Census block. Each household has a householder in the U.S. Census, and the Census provides the distributions of householders by sex and by household type. In this actual scenario, there would have to be one female householders (to the one house). In this hypothetical simulated outcome, the only result would be to have a female householder under the age of 10 years old.

This is not realistic; but more importantly it would be inconsistent with the data because all households in the Census have a householder above the age of 16 years. The options are to (1) choose with replacement and then invalidate other distributions or (2) choose without replacement and create the population exactly as it was in 2010. For subsequent distributions among the data
set to remain relevant, all distributions in the national model are sampled without replacement. This choice represents a significant deviation from previous iterations of this trip synthesizer, but is implemented in order to match reality as consistently as possible. There is no premium on randomness when the goal is to mimic reality. These methods allow Task 1 to build a synthetic population that more closely resembles the true population exactly.

2.2 Task 2: Workplace Assignment

Work assignment is the process by which each eligible adult is assigned to a place of work. This method is applied in three separate steps: (1) Assigning a County of Work, (2) Assigning an Industry of Work, and (3) Assigning a Place of Work. In each of the steps, a Gravity Model of attraction is used to evaluate the likelihood of a particular worker working in a county, an industry within that county, and a place of work within that industry within that county. First, the existing population must be distilled into those who would likely be a working resident. To recall, there are eight classes of workers, each a function of age and household type. Of those eight, only three qualify to be assigned a workplace in this model:

- Traveler Type 2: School-Work in County
- Traveler Type 4: College-Work in County
- Traveler Type 5: Typical Traveler

The rest are disqualified because they are retired, too young, do not travel to work today, or do not have a job.

While Census blocks are the primary unit for generating the populace, hereafter this project uses U.S. counties as the scope of analysis and for generating trip patterns. This aggregation is both necessary and appropriate. The data needed for business and school activities are not available at a level that is more local than the county. In addition, on average, the trips taken by individuals span one or several counties, but could span hundreds of census blocks, so counties represent a balance between localization and practicality. For a reference on the scale of counties to Census blocks, and counties within the U.S., refer to Section 2.7. While the New Jersey State Model employed a proprietary numbering system to uniquely identify the counties within New Jersey and surrounding areas, the need for a more robust method of identification is necessary to account for the extended geographical reach. This model uses Federal Information Processing Standards (FIPS) county codes - a five digit number - to uniquely identify each of the 3,143 counties or county equivalents within the
United States.\textsuperscript{5} The FIPS county code is relied upon as the unique identifier of county data, and is distinctly important in translating all of the data across multiple formats and tasks. Consequently, every data record for a resident is given geographic identifiers as well as an eleven digit code where the first two digits are the FIPS state code of his or her resident state, and the remaining nine are a pointer to the row in his or her home state residence file. The combination of these attributes permits easy methods to quickly find and sort residents and trips geographically.

The first step of Task 2 is to assign each worker a county of work, given his or her county of residence. To inform this assignment, Task 2 uses the Journey-to-Work Census, which details the flow of workers from home county to workplace county. For each county, a weighted distribution is assembled using a Gravity Model that measures the attractiveness of work counties for a given residence county. The Gravity Model is a way to ascribe a numeric value of attraction - a gravitational pull - to a location, which declines with distance squared and increases with the popularity of the destination. The model, outlined in Figure 2.2, creates the weights that comprise a distribution from which to draw a random work county. For each qualified worker within each residence county, Task 2 draws upon this distribution and assigns the worker a county of work.

\begin{align*}
\text{Let } & h \text{ be the index of the worker’s home county} \\
\text{Let } & J \text{ be the set of counties that are commuting destinations for workers in county } h \\
\text{Let } & x_{h,j} \text{ be the number of workers who commute from county } h \text{ to county } j, \text{ per the Journey-to-Work Census} \\
\text{Let } & D_{h,j} \text{ be the distance between the geographical centroids of county } h \text{ and county } j \\
\text{Let } & w_{h,j}, \text{ the weight of attraction to work in county } j \text{ if you live in county } h, \text{ be defined as:} \\
& w_{h,j} = \frac{x_{h,j}}{D_{h,j}} \frac{1}{\sum_j x_{h,j} D_{h,j}^2} \quad \forall \ j \in J
\end{align*}

Figure 2.2: The Gravity Model For Assigning County of Work

The second step of Task 2 is to assign each worker to an industry. This project uses the same North American Industry Classification System (NAICS) used by the U.S. Census Bureau to categorize 20 possible industries, named in Table 3.2 in Section 3.3. Each worker is given a work industry through assignment of one of 20 NAICS two-digit industry codes. The designation of an industry of work requires four inputs: gender, income, county of work, and the distribution of employment by gender by industry within the particular county of work. To create such a distribution, Task 2 uses a U.S. Census table titled “Industry by Sex and Median Earnings.”\textsuperscript{6} This data set relates industry

\textsuperscript{5}County equivalents within Louisiana are called ‘Parishes’, and in Alaska called ‘Boroughs’ or ‘Census Areas’
\textsuperscript{6}Industry Income Table
participation rates by gender and by the median income of workers within each industry for all counties in the United States. A full review of this data source can be found in Section 3.2.2. Again, using a Gravity Model, Task 2 creates a distribution relating a worker to the range of industries available within his or her county of work, which is outlined in Figure 2.3.

Let $i$ be the index of the worker’s work county
Let $\text{Inc}$ be the income of the worker
Let $k$ be one of the 20 NAISC industries in the set $K$
Let $\text{MedInc}_k$ be the median income of workers of the same gender in the $k$-th industry
Let $e_{i,k}$ be the number of employees in county $i$ of the same gender who work in industry $k$

Let $w_{i,k}$, the weight of attraction to industry $k$ for this particular worker, be defined as:

$$w_{i,k} = \frac{e_{i,k} (\text{Inc} - \text{MedInc}_k)^2}{\sum_k e_{i,k} (\text{Inc} - \text{MedInc}_k)^2} \quad \forall \ k \in K$$

Figure 2.3: The Gravity Model For Assigning Industry of Work

The third step of Task 2 is to assign each worker, given his or her county of work and industry of work, to a specific employer. Data were assembled for each state, or state equivalent, of the employers in the state. For a description of the data set used to identify a reasonably comprehensive list of all of the businesses in the nation, refer to Section 3.2.3. The county of workplace and industry of work qualifies a particular distribution of all of the possible employers in that industry in that county. Task 2 creates the distribution following the set of equations modeled in Figure 2.4, which is another version of a Gravity Model.

Let $i$ be the index of the worker’s work county
Let $k$ be the index of the worker’s industry ($k = 1, \ldots, 20$)
Let $J$ be the set of Employers within industry $K$ in county $i$
Let $e_{i,k,j}$ be the number of employees at workplace $j$, which is within county $i$ and in industry $k$
Let $D_{h,j}$ be the distance between the the home of the worker (located at the geographical centroid of his or her home Census block, $h$) to workplace $j$

Let $w_{h,j}$, the weight of attraction to workplace $j$ for this particular worker, be defined as:

$$w_{h,j} = \frac{e_{i,k,j}}{\sum_j e_{i,k,j}} \quad \forall \ j \in J$$

Figure 2.4: The Gravity Model For Assigning Place of Work

After assigning an employer to all workers within a state, Task 2 produces a revised file of residents that appends additional data about employment, or lack thereof, to the output from Task
1. Figure 2.5 presents a sample of one of the 51 output files from Task 2. Workplaces are essential anchors to the trip tours taken by most residents. The second essential anchor that applies to a large swathe of the population are schools, which is dealt with next.

### Task 2 Sample Output for West Virginia Residents

<table>
<thead>
<tr>
<th>Task 1</th>
<th>Work County</th>
<th>Work Industry</th>
<th>Employer</th>
<th>Work Address</th>
<th>Work City</th>
<th>Work State</th>
<th>Work County</th>
<th>Paid/Emp Ratio</th>
<th>Employees</th>
<th>Work Lat</th>
<th>Work Lon</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>Non-Worker</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>Non-Worker</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>54001</td>
<td>72</td>
<td>MEDALLION RESTAURANT</td>
<td>75 MAIN ST</td>
<td>PHILIPPI</td>
<td>WV</td>
<td>Barbour</td>
<td>2.33</td>
<td>10</td>
<td>29.1318</td>
<td>-80.0389</td>
<td></td>
</tr>
<tr>
<td>54001</td>
<td>71</td>
<td>SUCEDENINK COMMUNICATIONS</td>
<td>88 W 5TH ST</td>
<td>BUCKHANNON</td>
<td>WV</td>
<td>Upshur</td>
<td>1.86</td>
<td>100</td>
<td>38.9999</td>
<td>-80.2180</td>
<td></td>
</tr>
<tr>
<td>54031</td>
<td>62</td>
<td>BARNS &amp; NOBLE LLC</td>
<td>1000 UNIVERSITY CTR DR</td>
<td>MORGANTOWN</td>
<td>WV</td>
<td>Monongalia</td>
<td>4.50</td>
<td>50</td>
<td>39.8572</td>
<td>-80.0038</td>
<td></td>
</tr>
<tr>
<td>54031</td>
<td>62</td>
<td>UNITED HOSPITAL CTR INC</td>
<td>327 MEDICAL PARK DR</td>
<td>BRIDGEPORT</td>
<td>WV</td>
<td>Harrison</td>
<td>0</td>
<td>1890</td>
<td>39.2421</td>
<td>-80.2776</td>
<td></td>
</tr>
<tr>
<td>54001</td>
<td>31</td>
<td>ALENIA TIMBER CORP</td>
<td>INDUSTRIAL PARK RD</td>
<td>BURLINGTON</td>
<td>WV</td>
<td>Barbour</td>
<td>4.61</td>
<td>20</td>
<td>39.9977</td>
<td>-79.9933</td>
<td></td>
</tr>
<tr>
<td>54083</td>
<td>44</td>
<td>ROY'S RV SUPERCENTER</td>
<td>215 W 5TH ST</td>
<td>ELKINS</td>
<td>WV</td>
<td>Randolph</td>
<td>4.79</td>
<td>23</td>
<td>39.5254</td>
<td>-79.8577</td>
<td></td>
</tr>
<tr>
<td>54001</td>
<td>72</td>
<td>GAREFF'S RESTAURANT &amp; PUB</td>
<td>390 W WOODWIND MALL</td>
<td>BRIDGEPORT</td>
<td>WV</td>
<td>Harrison</td>
<td>4.82</td>
<td>45</td>
<td>39.3139</td>
<td>-80.2797</td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>Non-Worker</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>Non-Worker</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>54001</td>
<td>72</td>
<td>CAMP BARBOUR</td>
<td>4-H RD</td>
<td>PHILIPPI</td>
<td>WV</td>
<td>Barbour</td>
<td>4.82</td>
<td>10</td>
<td>29.1358</td>
<td>-80.0479</td>
<td></td>
</tr>
<tr>
<td>54001</td>
<td>31</td>
<td>SNYDER INDUSTRIES INC</td>
<td>34 MATTANGIO DR</td>
<td>PHILIPPI</td>
<td>WV</td>
<td>Barbour</td>
<td>2.15</td>
<td>15</td>
<td>29.1358</td>
<td>-80.0404</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.5: Task 2 Sample Output for West Virginia Residents**

### 2.3 Task 3: School Assignment

School assignment is the process by which each student, a resident aged six to 22 years and not living within institutional group quarters, is assigned to an appropriate school. The school assignment process works in a similar way as work assignment: a hierarchy of steps that narrow in on the right school. First, students are designated to attend a kindergarten, elementary or primary school, secondary school, or post-secondary school based on their age. Insufficient data on pre-kindergarten and nursery schools prevented assigning schools for young children below the age of six years. K-12 students are split between private and public schools, based on respective enrollment statistics within the United States. Students in post-secondary schools are designated to attend a four-year institution or graduate/professional program, a two-year associates degree program, or a non-degree granting program. The next step is selecting a county to which the student will travel for school. Given the county of school and the type of school, a distribution is built using the Gravity Model to select a particular school for the student. The model, which surveys all geographically adjacent counties for possibly places of schooling, is shown in Figure 2.6.\(^7\)

The restriction on the location of the school attended is done primarily for simplicity. However, this assumption makes sense intuitively. First, for post-secondary schools, often students live in dorms (indeed, a majority of post-secondary students attend four year colleges and live on campus), and so are considered living in group quarters, either near or on campus. Because of this proximity,

---

\(^7\)The U.S. Census provides a County-Adjacency list for all counties.
Let $h$ be the index of the student’s home county
Let $k$ be the type of school attended by the student (indicating public/private and elementary/middle/high/post-secondary)
Let $C$ be the set of counties, by index $c$, that are adjacent to $h$, the student’s home county
Let $x_{c,j}$ be the enrollment in school $j$, of type $k$, in county $c$
Let $D_{h,c}$ be the distance between the geographical centroids of county $h$ and county $c$
Let $w_{h,c,j}$, the weight of attraction to school $j$ in county $c$ if the student lives in county $h$, be defined as:

$$w_{h,c,j} = \frac{x_{c,j}}{D_{h,j}^2} \sum_j x_{c,j} \frac{1}{D_{h,j}^2} \forall \ c \in C$$

where the “distance” to your home county is taken to be $D_{hh} = 0.75 * \min_c \{D_{h,c}\}$

Figure 2.6: The Gravity Model For Assigning Schools

it is almost assuredly true that their school of choice is within their home county, or at the very least a neighboring one. If the Census designated a resident as living within a college dormitory group quarters, his or her county of school is automatically assigned to be his or her home county. For primary and secondary schools, with the possible exception of private schools, students are very likely to attend a school within their own township, let alone county. For private schools, the range might be extended, but neighboring counties most likely encompass the vast majority of private secondary schools attended for a particular county.

The data set for school enrollment required the most extrapolation and assumptions, because enrollment in post-secondary colleges is often unreported, and so unattainable on a national scale without manually searching for enrollment of specific schools. Names and locations of such schools is available, however. Because of these data constraints, a method of extrapolation was used to assign likely enrollment to post-secondary schools, which combined data for state enrollment in post-secondary educational institutions of all types with employee data. An exact description of this method is detailed in Section 3.3.

![Figure 2.7: Task 3 Sample Output for Tennessee Residents](image)
In Figure 2.7, a sample output of Task 3 is shown for random residents within Tennessee. The last three columns, which detail school information, are added onto the existing data set of simulated residents. Task 3 concludes the assignment of the rigid activities, or definitive anchors, of daily routine travel. The combined information simulated for residents will be used next to inform the rest of the trips that he or she will take during an average day.

2.4 Task 4: Tour Assignment and Activity Patterns

The next two tasks complete the daily trip tour. Before assigning individual points of travel along that tour, Task 4 assigns each resident a particular activity pattern, or chain of trip types for their simulated week day. The tours are combinations of Home (H), Work (W), School (S), and Other (O), that reflect all the possible trip tours that begin and end at home with a maximum of trip allowed. Given each resident’s Traveler Type, Task 4 selects an activity pattern from a predetermined distribution of tour types for that traveler type. The distribution of tour chains among a particular traveler type is designed so that the average number of trips taken reflects the true average number of trips taken daily (between three and four trips). The distribution is also constrained by the characteristics of the Traveler Type. For example, a non-worker traveler type is not allowed on an activity pattern that includes a workplace as a node of travel.

Table 2.2 details all of the 21 possible activity patterns of residents within the simulation. The distribution of activity patterns among Traveler Types is a modified version of Mufti’s original computation. His original solution enabled non-workers and non-students to travel to work or to school, respectively. This would be inconsistent with the traveler type designations, and previous Tasks, so a revised distribution of daily trip tours for each traveler type is shown in Table 3.3 in Section 3.4. In addition, two more activity patterns are added to include trip tours that did not involve a work place or school. While, these distributions are contrived, but are made to be reflective of data sources that detail the observed trip purposes and average number of trips taken in one day. Task 4 assigns each resident in the entire U.S. a daily trip tour, state by state. This activity pattern, or chain of individual trips, is assigned particular destinations for each leg of the trip tour in Task 5.

---

8Note. Figure 2.7 does not show all the range of data columns for each data record, but displays an abbreviated sample for presentation
Table 2.2: Daily Trip Tours Definitions

<table>
<thead>
<tr>
<th>Trip Tour</th>
<th>Representation</th>
<th>Trip Ends</th>
<th>Trip Tour</th>
<th>Representation</th>
<th>Trip Ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>0</td>
<td>11</td>
<td>H- W- H- O- H</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>H- W- H</td>
<td>2</td>
<td>12</td>
<td>H- S- H- O- H</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>H- S- H</td>
<td>2</td>
<td>13</td>
<td>H- O- H- O- H</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>H- O- H</td>
<td>2</td>
<td>14</td>
<td>H- W- O- W- H</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>H- W- S- H</td>
<td>3</td>
<td>16</td>
<td>H- S- O- H- O- H</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>H- S- O- H</td>
<td>3</td>
<td>18</td>
<td>H- S- H- O- O- H</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>H- W- S- O- H</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.5 Task 5: Trip Destination Assignment

Trip Destination Assignment is the assignment of each trip node on a daily tour to particular destinations, such as restaurants, gyms, and other errand locations. Task 5 reads in the generalized activity pattern of each resident and assigns each leg of the trip to an exact destination. Using the same Employee Patronage data used in Task 2, Task 5 selects points of attraction based on a Gravity Model identical to Task 2’s selection of employer, but substitutes rates of employment with rates of patronage.

The objective of Task 5 is to assign residents to their places of daily patronage, while sending enough patrons to each place of patronage so that the data align with the simulations as closely as possible. However, the patronage data are less reliable than the travel behavior as indicated in Census surveys, so the rates of travel are given precedence over rates of patronage. For example, if a store has 100 daily patrons, Task 5 does its best to send 100 trips there but will not send travelers on more trips than already prescribed to meet that 100 patron rate.

Previous Tasks provided the locations of all homes, places of work, and schools for every resident. What remains for Task 5 is to specify all of the “Other” trips taken by residents, which are the trips to places other than home, work, or school. A set of assumptions regarding the spatial distribution of other trips are considered in Task 5, namely:

- All Other trips are confined to geographically adjacent counties with respect to the county of trip origination. For example:
  - A trip tour that is Home-School-Other would consider all counties adjacent to the School County for the Other trip
  - A trip tour that is Home-Other would consider all counties adjacent to the Home County for the Other trip
- Other trips that go from work and back to work - considered ‘lunch trips’ - must be within
five miles of the place of work
• The minimum distance for Other trips is 0.5 miles to filter out any trips that might not use motorized transportation

Task 5 compiles all of the attributes of daily trip tours and outputs a master file of all the activities of each resident, sorted by his or her Person ID number (computed in Task 1). A sample output of Task 5 is shown in Figure 2.8 for a resident in Indiana. This particular resident, with an Activity Pattern of 19, leaves home for work at Buy-N-Save, then leaves work and goes to Port Cape Girardeau, a restaurant. Then, he or she goes to H&R Block to run an errand. In the second to last trip, he or she visits Chartwells, a retail store, before returning home to end his or her day. Node 8, or the final node in all trip chains, is always implicitly Home and so is not listed. The chain of trips taken by each resident is then passed to Task 6 which provides the final layer of disaggregation: time.

<table>
<thead>
<tr>
<th>Residence State</th>
<th>County Code</th>
<th>Tract Code</th>
<th>Block Code</th>
<th>HH ID</th>
<th>Person ID Number</th>
<th>Activity Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>003</td>
<td>957800</td>
<td>1034</td>
<td>27508</td>
<td>17000577380</td>
<td>19</td>
</tr>
<tr>
<td>Node 1 Type</td>
<td>Node 1 Predecessor</td>
<td>Node 3 Successor</td>
<td>Node 1 Name</td>
<td>Node 1 County</td>
<td>Node 1 Lat</td>
<td>Node 1 Lon</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>W</td>
<td>Home</td>
<td>170001</td>
<td>37.34286</td>
<td>-89.26400</td>
</tr>
<tr>
<td>Node 2 Type</td>
<td>Node 2 Predecessor</td>
<td>Node 2 Successor</td>
<td>Node 2 Name</td>
<td>Node 2 County</td>
<td>Node 2 Lat</td>
<td>Node 2 Lon</td>
</tr>
<tr>
<td>W</td>
<td>H</td>
<td>O</td>
<td>BUY-N-SAVE</td>
<td>17003</td>
<td>37.30819</td>
<td>-89.44815</td>
</tr>
<tr>
<td>Node 3 Type</td>
<td>Node 3 Predecessor</td>
<td>Node 3 Successor</td>
<td>Node 3 Name</td>
<td>Node 3 County</td>
<td>Node 3 Lat</td>
<td>Node 3 Lon</td>
</tr>
<tr>
<td>O</td>
<td>W</td>
<td>H</td>
<td>PORT CAPE GIRARDEAU</td>
<td>17002</td>
<td>37.30360</td>
<td>-89.51805</td>
</tr>
<tr>
<td>Node 4 Type</td>
<td>Node 4 Predecessor</td>
<td>Node 4 Successor</td>
<td>Node 4 Name</td>
<td>Node 4 County</td>
<td>Node 4 Lat</td>
<td>Node 4 Lon</td>
</tr>
<tr>
<td>H</td>
<td>O</td>
<td>O</td>
<td>Home</td>
<td>170000</td>
<td>37.34266</td>
<td>-89.26403</td>
</tr>
<tr>
<td>Node 5 Type</td>
<td>Node 5 Predecessor</td>
<td>Node 5 Successor</td>
<td>Node 5 Name</td>
<td>Node 5 County</td>
<td>Node 5 Lat</td>
<td>Node 5 Lon</td>
</tr>
<tr>
<td>O</td>
<td>H</td>
<td>H</td>
<td>H&amp;R BLOCK</td>
<td>17003</td>
<td>37.22104</td>
<td>-89.27382</td>
</tr>
<tr>
<td>Node 6 Type</td>
<td>Node 6 Predecessor</td>
<td>Node 6 Successor</td>
<td>Node 6 Name</td>
<td>Node 6 County</td>
<td>Node 6 Lat</td>
<td>Node 6 Lon</td>
</tr>
<tr>
<td>H</td>
<td>O</td>
<td>O</td>
<td>Home</td>
<td>170000</td>
<td>37.34266</td>
<td>-89.26403</td>
</tr>
<tr>
<td>Node 7 Type</td>
<td>Node 7 Predecessor</td>
<td>Node 7 Successor</td>
<td>Node 7 Name</td>
<td>Node 7 County</td>
<td>Node 7 Lat</td>
<td>Node 7 Lon</td>
</tr>
<tr>
<td>O</td>
<td>H</td>
<td>H</td>
<td>CHARTWELLS</td>
<td>17003</td>
<td>37.61959</td>
<td>-89.18487</td>
</tr>
</tbody>
</table>

Figure 2.8: Sample Output for Indiana Resident

2.5.1 Mode Split for Long-Distance Trips: Air Travel

The Journey-to-Work Census details worker flows that often span many states, and hundreds of miles, as well as international travel. This thesis considers long trips to be those trips which have an international destination or a commute length longer than 200 miles. School trips, which are geographically restricted to neighboring counties, and Other trips, which are similarly constrained, cannot be considered long trips. The objective of this thesis is to design a data set of all the trips taken by all residents that could be reasonably served by aTaxis. Trips that are longer than 200 miles are most likely trips taken by airplane, not by car - traditional automobile or aTaxi. For this reason, Task 5 introduces a simple mode split.

Task 5 identifies long trips and changes the activity pattern. The ‘commute’, which was once a
200+ mile trip, is altered so that it includes a trip from home to a nearby airport, a flight, and a trip from the end-point airport to the final destination. The airports are selected using a Gravity Model where the level of attraction to a particular airport increases with the land-area of the airport (its size) and declines with distance squared. Land-area is an unusual metric, but in the database of all airports provided by the F.A.A., the only data item that could be used as a proxy for airport popularity was its size. Logically, a large airport has many runways and terminals, and thus should be more attractive. Once the worker arrives at his or her work destination via multi-modal travel, the rest of their activity pattern follows the conceptual norms of typical business travel. Although this is a home-based model, these travelers end their day at a hotel near their work place instead. The hotel is selected just as a place of patronage is selected, but using a confined list of NAICS qualified hotels. Although the opportunity to send them on other trips exists, for simplicity, their activity pattern ends there.

<table>
<thead>
<tr>
<th>Residence State</th>
<th>County Code</th>
<th>Tract Code</th>
<th>Block Code</th>
<th>HH ID</th>
<th>Person ID Number</th>
<th>Activity Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>061</td>
<td>000202</td>
<td>2003</td>
<td>3339140</td>
<td>36095083402</td>
<td>14</td>
</tr>
</tbody>
</table>

Figure 2.9: Sample Output for Air Travel Mode Split

This mode split, while basic, is a proof of concept that multi-modal travel can be introduced into this model to improve the reality of certain trips. Figure 2.9 shows the result of this mode split for a Manhattan resident. This worker is traveling from the Lower East Side of Manhattan all the way to Los Angeles. It is far more likely he or she takes a plane rather than an aTaxi. However, aTaxis will serve the other legs of the trips, after mode split. His or her revised trip tour now includes a trip from Manhattan to John F. Kennedy International Airport in nearby Queens to take a cross-country flight to LAX. After work at American Honda Motor Co. Inc., he or she retires to the Courtyard hotel chain in Costa Mesa, CA. Note, his or her original trip pattern is code 14, which is H- W- O- W- H. This would have meant that he or she traveled cross-country twice, to

9Places of business with NAICS code 7211
work and back home, in an average work day. While this is possible, it is not likely. A far better representation of this worker's travels involves a business trip with air travel.

It is important to note that this mode split is conceptually identical to any mode split possible for this model. First, the trips that might reasonably travel on a different mode of transportation must be identified. This involves understanding the decision-making behind mode choice. Here the decision of the traveler who is faced with a trip in excess of 200 miles is to take a flight instead of driving. The model imitates this personal choice. Second, the trips must be translated to the network of mode options - air travel and aTaxi travel in this case. Again, discrete choices need to be simulated. If the additional distance necessary to travel from home to the airport and from an airport to work exceeds the distance of direct travel, a traveler might choose to drive instead of fly. The same structure can be applied to trips that might utilize trains or available mass transit.

2.6 Task 6: Arrival and Departure Time Assignment

Task 6 provides the temporal attributes of the daily trips that make up the trip tours built in Task 5. The objective of Task 6 is to schedule the activities within the day by giving each node of travel (Home/Work/School/Other) an arrival time, duration of stay, and departure time. The arrival time of the next node is the departure time of the previous node assuming a constant traveling speed of 30 miles per hour. The daily schedules, and the timing of activities, are organized to reflect the general trends of commute, school, and errand patterns throughout time as shown by Figure 3.1 in Section 3.6. To do so, Task 6 begins with assumptions that use NAICS industry codes to characterize employment start and end times and patronage stays.

For the first trips of the day - often to work or to school - the arrival time to that location is a random variable based on the bell time of the destination. Each arrival to work or to school is drawn from an exponential distribution with a rate parameter of five minutes. A window of arrival time is considered that begins 10 minutes before the set bell time. This scheme is equivalent to assuming that arrivals follow an exponential distribution where the expected arrival time is five minutes before the bell time, and approximately 10% of arrivals will be 'late'. Empirically, arrival times often follow exponential distributions. In addition, sequences of exponential arrivals to one location will follow a Poisson distribution where the scale parameter is the number of workers or students reporting to that location. Departure times from work and school follow a reverse process, where the window begins at the end bell time. Here, the expected departure time is five minutes.

\( ^{10} \)For tours that begin with an Other trip, the departure time from home to begin the tour is uniformly sampled from 9AM to 1PM
after the bell time. This assumes one is equally likely to arrive five minutes early to work as one is to leave five minutes after the end of a shift.

Each subsequent trip of the day - patronage trips, part-time work, part-time school, home trips - is given a duration that characterizes the likely time spent at each location. For patronage trips, duration of the visit is drawn from a normal distribution with an industry mean and a 15% variance. Similarly, for part-time work and part-time school, the duration is sampled from a normal distribution with a mean of three hours and a 15% variance. For home trips - a trip back home that is not the final trip of the day - the duration of stay is uniformly sampled from 15 minutes to one hour.

Task 6 begins with scheduling the first day, and then walks through the daily tour computing durations of stay at one node, the departure time from the node, and the arrival time to the next node, and so on. In this way, Task 6 produces a trip sequence file for each resident in the system which adds temporal attributes of their daily trip tour computed in Task 5. Task 6 schedules trips completely independently of all other travelers. For example, it does not aggregate first trip departure times by household. This is done for simplicity and because correlations between trips taken by family members are not known well enough. A sample output of Task 6 is shown in Figure 2.10 for a worker in Mississippi’s Grenada County. All times here are displayed in a 24-hour clock, but are actually stored as seconds from midnight. In addition, Node 7 and 8 data are not shown here because this trip tour ends for the day at Node 6.

![Figure 2.10: Sample Output for Mississippi Resident](image)

This sample output, chosen randomly, displays the complete day for a construction worker, who commutes roughly 30 minutes to work early in the morning, then goes to McDonald’s after work ends around 5:00PM, before heading home for a brief period. He makes a quick errand to a Dollar General store before retiring for the night at around 9:30PM. At the end of Task 6, all trips taken by
all residents are represented with reasonably distributed disaggregate spatial and temporal attributes for a given work day.

2.7 Geographical Assumptions of the Model

This methodology considers and uses several tiers of geographical data aggregation used by the Census. From largest to smallest, the geographical classifications used are states, counties, tracts, and blocks. In order to gain an appreciation for the varying geographic scopes of simulation, and their relative sizes, the geographical assumptions of this model are depicted here. The simulation is performed on 50 states and the District of Columbia. There are 3,143 county or county equivalents in those 51 state equivalents. Those counties are made up of, on average, 23 Census tracts. In turn, there are 151 Census blocks per tract on average. In Task 1, the model begins from the ground up with each of the 11,078,297 Census blocks. Each block is simulated independently and all of the data used, aside from household income, is at the block level. (U.S. Census Bureau, 2013)

In the next level up, Census tracts are the aggregation of several block groups. The average number of groups within a tract is approximately three. This geographical hierarchy contextualizes the application of tract level data to a block level simulation, as is the case for household income. While the decision was motivated by a data constraint, there is a significant degree of approximation when assuming a relatively larger area is uniformly representative of all of its parts. The assumption underlying the use of household income tract data is that income distributions for all blocks is uniform. This is not an empirically poor assumption in most instances, but one could imagine that in a Manhattan Census tract, for example, the make-up of the blocks contained within it is quite varied. To compare the sizes of the two geographical types, Figure 2.11 shows the city of San Francisco partitioned by Census tracts (black) and block groups (blue). Note, there are, on average, 50 blocks per block group.

From Task 2 onwards, the data are aggregated at the county level. As a result, county-level data are applied to all residents within the county. Again, this forfeiture of specificity is due to the lack of availability of more localized data. For example, all residents within a county, regardless of where they live within the county, are assumed to follow the same Journey-to-Work patterns. Of course, to compensate for this approximation, the gravity model distribution is inverse with distance squared. However, industry participation and school selection is applied uniformly to all residents within a county. Figure 2.12 demonstrates the relationship of counties to tracts, which contain on average 151 blocks.
Lastly, in instances of school enrollment by type and in forthcoming analysis, states are considered an aggregation of their respective counties and all respective sub-geographies. There is certainly a great degree of variation within each state, which often contain a range of urban and rural environments. Figure 2.13 shows the manner in which counties constitute states.
Figure 2.12: Relative Geography of Census Tracts Within Counties

Figure 2.13: Relative Geography of Counties Within Continental U.S.
Chapter 3

Data

This chapter presents the input data to the model that generates the characteristics of a simulated population and its personal travel. A simulation of this type will only ever be as good as its data. The synthesizer requires extensive demographic data, business activity, and school enrollment data for the U.S. to construct the profile of the average workday. Data sources are presented in the sequence that they are used by each Task of the model.

3.1 Task 1

Task 1 creates a file listing all of the residents, grouped by household, of each state or state equivalent, with associated attributes such as age, gender, and income. The data for this particular task is found in multiple sources from the U.S. Census.

3.1.1 2010 United States Census Summary File 1 Data

The demographic and housing data used in the synthesizer comes from the 2010 U.S. Census Summary File 1 (SF1). This data set contains the most comprehensive set of summary statistics compiled from the 2010 U.S. Census, with a particular focus on population items such as “sex, age, ..., household relationship, household size, family type, family size, and group quarters.” (SF 1 Technical Documentation, 2012, p. 1-1) This data, available at the block level, provided the synthesizer with the appropriate distributions so that it could accurately simulate and “populate” each census block within a state according to the levels in the year 2010.

While U.S. Census data are publicly available in a multitude of locations and formats, the raw data were obtained from the U.S. Census Bureau’s FTP site. The relevant data tables to this the-
sis are located in data segments 4, 5, 6 and 44. (SF 1 Technical Documentation, 2012) Through
the use of Microsoft Access macros and three separate SQL queries, three text files were created:
‘<ST>DemographicQuery’, ‘<ST>GroupQuartersQuery’, ‘<ST>FamilyQuery’ (where <ST>
represents the two character state abbreviation). Each row in each file represents a Census block within
that particular state. Table 3.1 describes the associated information contained within these text
files, which are inputs to Module 1, the programming component that executes Task 1.

<table>
<thead>
<tr>
<th>File Name</th>
<th>SF1 Data Segments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;ST&gt;DemographicQuery</td>
<td>4,44</td>
<td>Geographic Attributes; Sex by Age by Gender; Households by Size</td>
</tr>
<tr>
<td>&lt;ST&gt;GroupQuartersQuery</td>
<td>6</td>
<td>Population in Group Quarters by Group Quarters Type by Age</td>
</tr>
<tr>
<td>&lt;ST&gt;FamilyQuery</td>
<td>5</td>
<td>Population in Households by Household Type by Family Relation</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of Module 1 State Data Input Files

3.1.2 2010 American Community Survey of Household Income

The income data used in the synthesizer come from the 2010 American Community Survey
(ACS). While the U.S. Census is conducted every ten years, the ACS is a yearly sampling survey
intended to provide on-going, current estimates on many topics, particularly household income.
(ACS Technical Documentation, p. 4) The ACS is meant to supplement the decennial census. By
sampling from the population, the ACS is able to ask more detailed questions that cover additional
areas of interest. The ACS data is available at 1-Year, 3-Year, and 5-Year estimates that offer
users a trade-off between more current data and margin of error. While ACS data are available in
a multitude of formats, this study obtained the data through the use of the American FactFinder
database.

This project uses the data table titled ‘2008-2012 Household Income in the Past 12 Months (in
2012 Inflation-Adjusted Dollars)’, which are the 5-Year estimates of household income, by household
type for each census tract within each state or state equivalent.\(^1\) This data set is used to assign
household income, and then each individual, a personal income amount. Unfortunately, income
data are not available at the Census block level, so the assumption is made that income levels and
distributions are constant within each census tract. This project does not use the available margins
of error available for each estimate in the data set. The margins of error are generally small, and

\(^1\)The technical table name is S1910
“the estimates were used as weights to draw from, that is as a discrete distribution, and not to fit any sort of closed-form probability distribution around each value or a set of values.” (Mufti, 2012, p. 49)

3.2 Task 2

Task 2, and its programming parallel Module 2, assign each worker a specific place of employment by way of two intermediate steps. First it assigns each worker a county of work, then an industry of work within that county. The data for Task 2 is a combination of large scale public and private employment data.

3.2.1 2010 American Community Survey Journey-to-Work Census

In order to assign each worker to a county of work, this project drew from the U.S. Census’ commuting and employment data sets, particularly, ‘Residence County to Workplace County Flows for the United States and Puerto Rico Sorted by Residence Geography: 2006 - 2010’. (U.S. Census Bureau Metropolitan and Micropolitan: Journey-to-Work, 2010) For each U.S. county, this table lists all the counties to which its residents commute and the estimate for the number of commuters to each county. Because the data set included Puerto Rico and other U.S. Territories, it is modified and truncated before use to only include the 50 states, plus the District of Columbia. These data were obtained from the “Metropolitan and Micropolitan” research area of the United States Census Bureau and are instrumental in recreating the true spatial distribution between work and residence in the simulated population. A margin of error for the estimates is included in the data set, but is not used in the synthesis.

3.2.2 2010 American Community Survey Industry Type Participation by Gender and Median Income

The 2010 ACS provides detailed data sets by county describing the rates of participation in each category of industry (by NAICS code) by gender. Additionally, it expresses the median income of workers in each industry by county. This information is used to assign each worker to a particular industry, given their gender, income level, and workplace county.

This project uses the data table titled ‘Industry by Sex and Median Earnings in the Past 12 Months (in 2012 Inflation-Adjusted Dollars) for the Civilian Employed Population 16 Years and
Over,” particularly, the 2012 5-Year estimates.\(^2\) As with the ACS Income data set, margins of error are provided but not used by this thesis for the same reason explained in Section 3.1.2. In addition, as with all data from the ACS used by this thesis, this data set was obtained through searching through the American FactFinder database.

Because of the incomplete information on all employers within each county in the United States, adjustments to the distributions informed by this data set are made.\(^3\) This project, unfortunately, does not use a 100% comprehensive list of all employers within the United States. The distributions of participation within a particular industry detailed by the Census are assumed to reflect the true range and distribution of employment opportunities by industry within the United States. In rare instances, a discrepancy necessitates a compromise in the data used. For example, in County XYZ, there might not be any employers in Mining, Quarrying, and Oil and Gas Extraction in the list of employers used by this project. Again, this is due to the fact that a complete list of employers is not available. However, there might indeed be Mining, Quarrying, and Oil and Gas Extraction jobs there in reality, and the Census data would reflect this. In turn, the Mining, Quarrying, and Oil and Gas Extraction industry would have non-zero weight in the distribution used to select a work industry in that particular county for a given worker. If it were selected as the industry of work for a particular worker, there would be no available employer for that worker. The available solutions are to either make up a new employer or to prevent Module 2 from selecting industries for which there are no employers in a county. The latter option is chosen, because it is too far-fetched to create new employers and their associated attributes. The trade-off is that in certain, albeit rare, cases the distribution of work industry participation is altered and does not accurately reflect true industry participation rates as described by the U.S. Census.

### 3.2.3 Employee and Patronage Data

The Employee and Patronage data set contains a quasi-comprehensive list of businesses within the United States up to 2012 with detailed information about each business, such as its location and industry type as well as the number of employees, patrons, and sales volume. This information was obtained from the 2012 InfoGroup’s 2012 U.S. Businesses Database. InfoGroup is a data and marketing company that focuses on generating sales leads and packaging information for sale to businesses. This large database contained 30 CSV files with approximately 500,000 entries each. The work done to identify the database, and create 51 files for each state and state equivalent

\(^2\)The technical table name is S2403

\(^3\)Section 3.2.3
containing all of the business listings within each state, is attributed to Luke Cheng and Judy Sun, as part of their work for the ORF467-2013. For easy use, the files are further partitioned into county business lists.

The data set contains issues with respect to the ratios of Employees to Patrons. The business data from the New Jersey State Model were obtained from a private source, and businesses were manually assigned Employee-Patronage ratios, based on industry and inspection. Manual efforts are not available at such a large scale, so to assign ratios to the business listings for the entire nation, a combination of inputs are used. The New Jersey State Model’s business data contained fine-tuned ratios. Each business included fields that listed its industry, by NAICS code, and its general Employee Size Range. The national business data contain identical fields, so to obtain ratios for the national file, an average of the New Jersey employment data ratios was taken for each NAICS code and Employee Size Range combination. The ratios, already vetted within New Jersey, were then applied to each business in the national data set, given their NAICS code and Employee Size Range.

<table>
<thead>
<tr>
<th>NAICS Category Code</th>
<th>NAICS Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Agriculture, Forestry, Fishing and Hunting</td>
</tr>
<tr>
<td>21</td>
<td>Mining, Quarrying, and Oil and Gas Extraction</td>
</tr>
<tr>
<td>22</td>
<td>Utilities</td>
</tr>
<tr>
<td>23</td>
<td>Construction</td>
</tr>
<tr>
<td>31-33</td>
<td>Manufacturing</td>
</tr>
<tr>
<td>42</td>
<td>Wholesale Trade</td>
</tr>
<tr>
<td>44-45</td>
<td>Retail Trade</td>
</tr>
<tr>
<td>48-49</td>
<td>Transportation and Warehousing</td>
</tr>
<tr>
<td>51</td>
<td>Information</td>
</tr>
<tr>
<td>52</td>
<td>Finance and Insurance</td>
</tr>
<tr>
<td>53</td>
<td>Real Estate and Rental and Leasing</td>
</tr>
<tr>
<td>54</td>
<td>Professional, Scientific, and Technical Services</td>
</tr>
<tr>
<td>55</td>
<td>Management of Companies and Enterprises</td>
</tr>
<tr>
<td>56</td>
<td>Administrative and Support and Waste Management</td>
</tr>
<tr>
<td>61</td>
<td>Education Services</td>
</tr>
<tr>
<td>62</td>
<td>Health Care and Social Assistance</td>
</tr>
<tr>
<td>71</td>
<td>Arts, Entertainment, and Recreation</td>
</tr>
<tr>
<td>72</td>
<td>Accommodation and Food Services</td>
</tr>
<tr>
<td>81</td>
<td>Other Services (Except Public Administration)</td>
</tr>
<tr>
<td>92</td>
<td>Public Administration</td>
</tr>
</tbody>
</table>

Table 3.2: NAICS Industries Used
3.3 Task 3

3.3.1 National Center for Education Statistics

To assign each eligible student to an appropriate school, data was obtained from the United States Department of Education for each listed primary and secondary school within the U.S. Particularly, a subsidiary of the D.O.E., the National Center for Education Statistics (NCES) maintains a “Common Core of Data” for each school that details its location, type (public/private, elementary/middle/high), and enrollment. To partition students into private and public schools, the national rates provided by the NCES for participation in each type of school for each age level are used.

3.3.2 Data for Post-Secondary Schools

The data for post-secondary schools (colleges, universities, non-degree granting schools) are less readily available and comprehensive than those for primary and secondary schools. While the names and locations are easily obtainable, enrollment is either not divulged by schools or is often incorrect. The NCES does not provide the level of detail for post-secondary schools as it does for secondary schools, because many fall under different accrediting categories and as such are not under the Department of Education’s direct purview. For this reason, a back-door solution to enrollment in specific colleges is used. It is assumed that for a post-secondary school to be eligible within the system, it would have workers (professors, administrators, staff) going there. For this reason, the school and its location would be in the Employee and Patronage files. Post-secondary schools are located within the Employee and Patronage data and made eligible for post-secondary students using an approximation.

Three types of post-secondary schools were considered because the level of detail provided by the NAICS Educational Services coding system only has three categories available within the Employee and Patronage files. The categories are: (1) Junior Colleges, Community Colleges, Associate-degree granting, (2) 4-year Baccalaureate programs and Graduate/Professional programs, and (3) Non-degree granting institutions. These three types match the U.S. Census’ categorization of the post-secondary educational programs: (1) 2-year, (2) 4-year or graduate, and (3) Non-degree granting. State level enrollment within each of these categories is available from the U.S. Census for 2010. The number of employees at each institution is used as a weight to distribute specific student enrollment to each post-secondary school within each state. In a sense, it is not unlike assuming a constant
state-level student-faculty ratio and applying it to each school to make sure total enrollment in each of the three categories matches summary data known at the state level.

### 3.4 Task 4

The distribution of activity patterns among traveler types is a modified version of the initial distribution created by Mufti, and used again by Gao. The driving data behind the distribution of activity patterns are the average number of trips taken each day and the distribution of trips by trip purpose. In 2009, the average number of trips taken per day per person was 3.79. (U.S. Department of Transportation, 2009) According to the 2009 National Household Transportation Survey, as a percentage of the average 3.79 trips, 10% of trips are to/from school or church, 15% of trips are to/from work, and 75% of trips are personal/errands/other. (U.S. Department of Transportation, 2009) The distributions are made to be approximations of this general empirical distribution. The distribution used is also a reconciliation of the traveler type category to possible activities. For example, if the resident is a non-student or non-worker, he or she cannot participate in activity patterns that involve a school or place of work, respectively. The table of activity pattern distributions is presented in Table 3.3 for each Traveler Type category.

### 3.5 Task 5

Task 5, which assigns all Other trips, uses the exact same data sources as Task 2, but extracts the patronage ratios instead of the employment ratios.

### 3.6 Task 6

The data behind Task 6 is the distribution of trip start time times throughout the day, in addition to timing rules and models for trip anchors such as school and work. The 2009 National Household Transportation Survey provides an example of the temporal distribution of trips throughout the day, by trip purpose in Figure 3.1. This distribution serves as the model target for its temporal assignments.

Guesswork was used to match NAICS codes to work start and end times as well as patronage duration so that the resulting distributions resemble the real ones. The parameters used in the model are shown in Table 3.4. The methodology of Task 6 details the assumptions made to achieve this
<table>
<thead>
<tr>
<th>Traveler Type</th>
<th>Do-Not-Travel</th>
<th>School-No-Work</th>
<th>School-Work</th>
<th>College</th>
<th>College-Work</th>
<th>Typical Worker</th>
<th>Home-Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tour Representation</td>
<td>H</td>
<td>1.0000</td>
<td>0.0100</td>
<td>0.0100</td>
<td>0.0050</td>
<td>0.0050</td>
<td>0.0040</td>
</tr>
<tr>
<td></td>
<td>H- W- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0075</td>
<td>0.0500</td>
</tr>
<tr>
<td></td>
<td>H- S- H</td>
<td>0.0000</td>
<td>0.1250</td>
<td>0.0500</td>
<td>0.1500</td>
<td>0.0075</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- S- W- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.4050</td>
<td>0.0000</td>
<td>0.2000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- S- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0750</td>
<td>0.1960</td>
</tr>
<tr>
<td></td>
<td>H- S- O- H</td>
<td>0.0000</td>
<td>0.3500</td>
<td>0.0850</td>
<td>0.3000</td>
<td>0.0075</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- O- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0750</td>
<td>0.3500</td>
</tr>
<tr>
<td></td>
<td>H- S- W- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.4500</td>
<td>0.0000</td>
<td>0.2600</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- S- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.2600</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0750</td>
<td>0.1500</td>
</tr>
<tr>
<td></td>
<td>H- S- H- O- H</td>
<td>0.0000</td>
<td>0.3250</td>
<td>0.0000</td>
<td>0.4500</td>
<td>0.0075</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- O- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- O- W- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0750</td>
<td>0.1500</td>
</tr>
<tr>
<td></td>
<td>H- W- O- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0750</td>
<td>0.1500</td>
</tr>
<tr>
<td></td>
<td>H- S- O- H- O- H</td>
<td>0.0000</td>
<td>0.1500</td>
<td>0.0000</td>
<td>0.0450</td>
<td>0.0075</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- H- O- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0050</td>
<td>0.1500</td>
</tr>
<tr>
<td></td>
<td>H- S- H- O- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- O- H- O- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1500</td>
</tr>
<tr>
<td></td>
<td>H- S- O- H- O- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>H- W- O- H- O- H- O- H- O- H</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>Sum</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>Expected No. of Trips</td>
<td>0.00</td>
<td>3.58</td>
<td>3.37</td>
<td>3.58</td>
<td>3.59</td>
<td>4.59</td>
<td>2.55</td>
</tr>
</tbody>
</table>

Table 3.3: Trip Tour Distributions

Figure 3.1: Empirical Distribution of Daily Trips by Start Time
result, but no concrete data beyond this are used directly. Usually confined to small sample surveys, exact transportation sequencing in time is not available at a large scale. There does not exist data about the exact timing of trip chaining on a disaggregate basis that are comprehensive enough to apply to general populations. As such, the guesswork here is meant to provide a framework for fine-tuning and continued modeling to push the simulation towards the empirical result shown in Figure 3.1.

<table>
<thead>
<tr>
<th>Industry</th>
<th>Employee Times</th>
<th>Patronage Times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Start Bell</td>
<td>End Bell</td>
</tr>
<tr>
<td>Agriculture, Forestry, etc.</td>
<td>7:30AM</td>
<td>4:30PM</td>
</tr>
<tr>
<td>Mining, Quarrying, etc.</td>
<td>7:30AM</td>
<td>4:30PM</td>
</tr>
<tr>
<td>Utilities</td>
<td>8:30AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Construction</td>
<td>7:30AM</td>
<td>4:30PM</td>
</tr>
<tr>
<td>Manufacturing</td>
<td>8:00AM</td>
<td>4:30PM</td>
</tr>
<tr>
<td>Wholesale Trade</td>
<td>9:00AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Retail Trade</td>
<td>9:00AM</td>
<td>6:00PM</td>
</tr>
<tr>
<td>Transportation and Warehousing</td>
<td>8:00AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Information</td>
<td>10:00AM</td>
<td>7:00PM</td>
</tr>
<tr>
<td>Finance and Insurance</td>
<td>9:00AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Real Estate, etc.</td>
<td>9:00AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Professional Services</td>
<td>9:00AM</td>
<td>6:30PM</td>
</tr>
<tr>
<td>Management of Companies, etc.</td>
<td>9:00AM</td>
<td>6:00PM</td>
</tr>
<tr>
<td>Administrative Services, etc.</td>
<td>8:30AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Education Services</td>
<td>8:30AM</td>
<td>4:00AM</td>
</tr>
<tr>
<td>Health Care, etc.</td>
<td>8:30AM</td>
<td>4:30AM</td>
</tr>
<tr>
<td>Arts, Entertainment, Recreation</td>
<td>11:00AM</td>
<td>8:00PM</td>
</tr>
<tr>
<td>Accommodation and Food Services</td>
<td>11:00AM</td>
<td>8:00PM</td>
</tr>
<tr>
<td>Other Services</td>
<td>9:00AM</td>
<td>5:00PM</td>
</tr>
<tr>
<td>Public Administration</td>
<td>9:00AM</td>
<td>5:00PM</td>
</tr>
</tbody>
</table>

Table 3.4: Bell Times and Durations by Industry
Chapter 4

Synthesizer Analysis and Results

4.1 Demographic Characteristics of Simulated Data

Population generation is a formula that is applied uniformly to each state. The total simulated population of the country is 308,745,538, the same as it was in 2010. The randomness in Task 1 is necessitated by aggregate statistics within the Census, such as age distributions and income. Consequently, the comparisons of the synthetic and real populations with respect to age and income are analyzed. A popular demographic analysis tool of the U.S. is the age pyramid, which breaks down the population by gender by age. In Figure 4.1 and Figure 4.2, the simulated national populations of each gender are compared to the population as it was in 2010.. The average absolute percent difference between each respective age bracket in the simulated data and national data is 2.96% and 2.78% for male and females, respectively.

To demonstrate the success and viability of Task 1 at state level, as well as the effects of applying tract level data to Census blocks, summary demographic statistics of South Carolina are presented in Table 4.1.¹ As expected, the simulated population in 2010 matches the population exactly as it was in 2010. One of the difficulties of this model is its reliance on 2010 data. In order to generally assess its applicability to the present, the 2012 population estimates are compared to the 2010 results. This, besides being a measure of population growth, is illustrative of the constraint imposed by using Census data published once every ten years.

The assignment of household income and personal income is extremely important because of its use in assigning industries of work. The household income statistics in Table 4.1 show the success of

¹The population counts for 2010 are taken from the U.S. Census. The statistics regarding households are taken from the 2008 - 2012 ACS Survey 5-Year Estimates
household income simulation. The limitation posed by applying Census tract data to its constituent blocks does not have a detrimental effect at all. Figure 4.3 shows the cumulative distribution of household income from which the simulation draws household income. The resulting cumulative distribution of household income demonstrates the closeness of income simulation on the whole spectrum of income levels. Note, the actual (non-simulated) data presented only show the total number of households within each bucket, and not the actual distribution within each bucket. Recall incomes within each bucket were sampled uniformly at random in the simulation. One income range: $100,000 to $149,999, is not exact. This discrepancy is due to the combining of the two income brackets, $100,000-$124,999 and $125,000-$149,999 within the simulation. To compare the
The demographic results of South Carolina are indicative of this model’s power: the simulated population matches the true population in both an aggregate and disaggregate fashion. While this
is only one state in the Union, there is nothing special about South Carolina. The model generically rebuilds each Census block within each state, so one can be sure that the success of South Carolina’s simulated population is indicative of the likely similarity between the 308,745,538 residents and their 308,745,538 simulated counterparts.

4.2 Home to Work Trips

Approximately 15% of the trips taken by this population are work commutes. Examined next is the nature of these commutes in space. Table 4.2 presents a statistical summary of all the distributions at the state level, and the following graphs present the cumulative distribution of trip length for each of the 51 regions of simulation. These graphs convey the statistical profile of the synthesized commutes for all workers at the state level, and follow curvatures one should expect from the Gravity Model used. In addition, commute length is a result of employer selection, and so is a function of the density of employment opportunities within the state. The analysis of commutes is meant to convey both the characteristics of commute trips themselves and their spatial distribution as synthesized by the model.

\[2\] The trips are classified by the residence state of the traveler, not the destination of the work trip.
Figure 4.5: Cumulative Distribution of Trip Length For Commutes by State (AL - CO)

(a) Alabama, Alaska, Arizona

(b) Arkansas, California, Colorado

Figure 4.6: Cumulative Distribution of Trip Length For Commutes by State (CT - HI)

(a) Connecticut, D.C., Delaware

(b) Florida, Georgia, Hawaii

Figure 4.7: Cumulative Distribution of Trip Length For Commutes by State (ID - KY)

(a) Idaho, Illinois, Indiana

(b) Iowa, Kansas, Kentucky
Figure 4.8: Cumulative Distribution of Trip Length For Commutes by State (LA - MN)

(a) Louisiana, Maine, Maryland

(b) Massachusetts, Michigan, Minnesota

Figure 4.9: Cumulative Distribution of Trip Length For Commutes by State (MS - NH)

(a) Mississippi, Missouri, Montana

(b) Nebraska, Nevada, New Hampshire

Figure 4.10: Cumulative Distribution of Trip Length For Commutes by State (NJ- OH)

(a) New Jersey, New Mexico, New York

(b) North Carolina, North Dakota, Ohio
Figure 4.11: Cumulative Distribution of Trip Length For Commutes by State (OK - SD)

(a) Oklahoma, Oregon, Pennsylvania

(b) Rhode Island, South Carolina, South Dakota

Figure 4.12: Cumulative Distribution of Trip Length For Commutes by State (TN - WA)

(a) Tennessee, Texas, Utah

(b) Vermont, Virginia, Washington

Figure 4.13: Cumulative Distribution of Trip Length For Commutes by State (WV - WY)

(a) West Virginia, Wisconsin, Wyoming
<table>
<thead>
<tr>
<th>State</th>
<th>% &lt;1mi</th>
<th>% &gt;60mi.</th>
<th>Mean Length</th>
<th>10%</th>
<th>Median Length</th>
<th>90%</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>2.00%</td>
<td>1.90%</td>
<td>14.99</td>
<td>2.77</td>
<td>10.40</td>
<td>30.04</td>
<td>4.5a</td>
</tr>
<tr>
<td>AK</td>
<td>6.05%</td>
<td>9.92%</td>
<td>21.23</td>
<td>1.49</td>
<td>7.28</td>
<td>59.80</td>
<td>4.5a</td>
</tr>
<tr>
<td>AZ</td>
<td>0.85%</td>
<td>5.57%</td>
<td>23.85</td>
<td>5.12</td>
<td>18.10</td>
<td>44.50</td>
<td>4.5a</td>
</tr>
<tr>
<td>AR</td>
<td>4.12%</td>
<td>1.56%</td>
<td>13.02</td>
<td>1.88</td>
<td>8.84</td>
<td>25.13</td>
<td>4.5b</td>
</tr>
<tr>
<td>CA</td>
<td>1.03%</td>
<td>2.50%</td>
<td>16.84</td>
<td>4.25</td>
<td>12.67</td>
<td>31.42</td>
<td>4.5b</td>
</tr>
<tr>
<td>CO</td>
<td>2.70%</td>
<td>1.10%</td>
<td>12.93</td>
<td>2.50</td>
<td>9.42</td>
<td>26.11</td>
<td>4.5b</td>
</tr>
<tr>
<td>CT</td>
<td>1.02%</td>
<td>1.27%</td>
<td>15.68</td>
<td>3.77</td>
<td>12.36</td>
<td>30.24</td>
<td>4.6a</td>
</tr>
<tr>
<td>DC</td>
<td>4.02%</td>
<td>0.40%</td>
<td>5.46</td>
<td>1.30</td>
<td>3.69</td>
<td>9.86</td>
<td>4.6a</td>
</tr>
<tr>
<td>DE</td>
<td>1.49%</td>
<td>2.10%</td>
<td>14.66</td>
<td>3.09</td>
<td>10.32</td>
<td>30.12</td>
<td>4.6a</td>
</tr>
<tr>
<td>FL</td>
<td>1.08%</td>
<td>1.35%</td>
<td>14.84</td>
<td>3.62</td>
<td>11.37</td>
<td>27.35</td>
<td>4.6b</td>
</tr>
<tr>
<td>GA</td>
<td>2.59%</td>
<td>1.27%</td>
<td>13.16</td>
<td>2.41</td>
<td>8.96</td>
<td>26.35</td>
<td>4.6b</td>
</tr>
<tr>
<td>HI</td>
<td>1.97%</td>
<td>2.64%</td>
<td>19.43</td>
<td>3.78</td>
<td>15.57</td>
<td>33.00</td>
<td>4.6b</td>
</tr>
<tr>
<td>ID</td>
<td>6.14%</td>
<td>1.63%</td>
<td>11.84</td>
<td>1.44</td>
<td>7.44</td>
<td>24.85</td>
<td>4.7a</td>
</tr>
<tr>
<td>IL</td>
<td>2.48%</td>
<td>0.93%</td>
<td>14.27</td>
<td>2.86</td>
<td>10.90</td>
<td>27.80</td>
<td>4.7a</td>
</tr>
<tr>
<td>IN</td>
<td>3.76%</td>
<td>1.16%</td>
<td>11.73</td>
<td>1.99</td>
<td>8.13</td>
<td>24.58</td>
<td>4.7a</td>
</tr>
<tr>
<td>IA</td>
<td>7.03%</td>
<td>1.20%</td>
<td>11.66</td>
<td>1.35</td>
<td>8.19</td>
<td>24.80</td>
<td>4.7b</td>
</tr>
<tr>
<td>KS</td>
<td>7.47%</td>
<td>1.16%</td>
<td>11.20</td>
<td>1.31</td>
<td>8.14</td>
<td>22.62</td>
<td>4.7b</td>
</tr>
<tr>
<td>KY</td>
<td>3.87%</td>
<td>1.39%</td>
<td>11.21</td>
<td>1.89</td>
<td>7.32</td>
<td>23.23</td>
<td>4.7b</td>
</tr>
<tr>
<td>LA</td>
<td>2.88%</td>
<td>2.27%</td>
<td>14.04</td>
<td>2.27</td>
<td>8.95</td>
<td>29.08</td>
<td>4.8a</td>
</tr>
<tr>
<td>ME</td>
<td>2.03%</td>
<td>3.94%</td>
<td>21.06</td>
<td>3.52</td>
<td>15.55</td>
<td>43.97</td>
<td>4.8a</td>
</tr>
<tr>
<td>MD</td>
<td>1.58%</td>
<td>0.79%</td>
<td>13.19</td>
<td>2.97</td>
<td>10.23</td>
<td>26.29</td>
<td>4.8a</td>
</tr>
<tr>
<td>MA</td>
<td>1.38%</td>
<td>0.89%</td>
<td>14.80</td>
<td>3.25</td>
<td>12.30</td>
<td>27.85</td>
<td>4.8b</td>
</tr>
<tr>
<td>MI</td>
<td>1.62%</td>
<td>1.32%</td>
<td>14.80</td>
<td>3.33</td>
<td>11.30</td>
<td>28.78</td>
<td>4.8b</td>
</tr>
<tr>
<td>MN</td>
<td>2.91%</td>
<td>1.84%</td>
<td>15.34</td>
<td>2.83</td>
<td>11.08</td>
<td>31.20</td>
<td>4.8b</td>
</tr>
<tr>
<td>MS</td>
<td>3.76%</td>
<td>2.54%</td>
<td>14.74</td>
<td>2.01</td>
<td>9.71</td>
<td>30.48</td>
<td>4.9a</td>
</tr>
<tr>
<td>MO</td>
<td>3.32%</td>
<td>1.32%</td>
<td>13.14</td>
<td>2.35</td>
<td>9.78</td>
<td>25.51</td>
<td>4.9a</td>
</tr>
<tr>
<td>MT</td>
<td>9.60%</td>
<td>2.53%</td>
<td>13.94</td>
<td>1.03</td>
<td>7.20</td>
<td>34.43</td>
<td>4.9a</td>
</tr>
<tr>
<td>NE</td>
<td>8.41%</td>
<td>0.91%</td>
<td>9.89</td>
<td>1.17</td>
<td>6.49</td>
<td>20.55</td>
<td>4.9b</td>
</tr>
<tr>
<td>NV</td>
<td>1.86%</td>
<td>3.27%</td>
<td>13.44</td>
<td>2.85</td>
<td>8.82</td>
<td>19.23</td>
<td>4.9b</td>
</tr>
<tr>
<td>NH</td>
<td>1.61%</td>
<td>1.98%</td>
<td>18.97</td>
<td>4.19</td>
<td>15.04</td>
<td>37.57</td>
<td>4.9b</td>
</tr>
<tr>
<td>NJ</td>
<td>1.40%</td>
<td>0.99%</td>
<td>12.97</td>
<td>3.05</td>
<td>9.30</td>
<td>27.19</td>
<td>4.10a</td>
</tr>
<tr>
<td>NM</td>
<td>4.52%</td>
<td>3.12%</td>
<td>14.57</td>
<td>1.71</td>
<td>7.81</td>
<td>35.98</td>
<td>4.10a</td>
</tr>
<tr>
<td>NY</td>
<td>2.72%</td>
<td>1.00%</td>
<td>12.08</td>
<td>2.15</td>
<td>7.94</td>
<td>26.44</td>
<td>4.10a</td>
</tr>
<tr>
<td>NC</td>
<td>1.95%</td>
<td>1.46%</td>
<td>13.27</td>
<td>2.70</td>
<td>9.30</td>
<td>26.39</td>
<td>4.10b</td>
</tr>
<tr>
<td>ND</td>
<td>11.10%</td>
<td>2.09%</td>
<td>12.93</td>
<td>0.09</td>
<td>6.12</td>
<td>28.52</td>
<td>4.10b</td>
</tr>
<tr>
<td>OH</td>
<td>2.25%</td>
<td>1.05%</td>
<td>12.56</td>
<td>2.69</td>
<td>9.28</td>
<td>24.98</td>
<td>4.10b</td>
</tr>
<tr>
<td>OK</td>
<td>4.34%</td>
<td>1.77%</td>
<td>13.71</td>
<td>1.96</td>
<td>9.74</td>
<td>27.71</td>
<td>4.11a</td>
</tr>
<tr>
<td>OR</td>
<td>3.21%</td>
<td>1.40%</td>
<td>13.40</td>
<td>2.24</td>
<td>10.34</td>
<td>25.57</td>
<td>4.11a</td>
</tr>
<tr>
<td>PA</td>
<td>2.08%</td>
<td>1.67%</td>
<td>13.65</td>
<td>2.74</td>
<td>9.72</td>
<td>26.82</td>
<td>4.11a</td>
</tr>
<tr>
<td>RI</td>
<td>1.93%</td>
<td>0.84%</td>
<td>12.05</td>
<td>2.61</td>
<td>9.05</td>
<td>24.08</td>
<td>4.11b</td>
</tr>
<tr>
<td>SC</td>
<td>2.05%</td>
<td>1.58%</td>
<td>14.04</td>
<td>2.86</td>
<td>10.43</td>
<td>26.81</td>
<td>4.11b</td>
</tr>
<tr>
<td>SD</td>
<td>10.38%</td>
<td>1.79%</td>
<td>12.10</td>
<td>0.97</td>
<td>6.75</td>
<td>27.20</td>
<td>4.11b</td>
</tr>
<tr>
<td>TN</td>
<td>2.19%</td>
<td>1.11%</td>
<td>12.45</td>
<td>2.56</td>
<td>9.02</td>
<td>25.07</td>
<td>4.12a</td>
</tr>
<tr>
<td>TX</td>
<td>1.56%</td>
<td>1.39%</td>
<td>14.96</td>
<td>3.74</td>
<td>12.18</td>
<td>26.03</td>
<td>4.12a</td>
</tr>
<tr>
<td>UT</td>
<td>3.23%</td>
<td>1.33%</td>
<td>12.15</td>
<td>2.19</td>
<td>7.97</td>
<td>24.46</td>
<td>4.12a</td>
</tr>
<tr>
<td>VT</td>
<td>3.73%</td>
<td>1.75%</td>
<td>15.88</td>
<td>2.42</td>
<td>11.85</td>
<td>32.47</td>
<td>4.12b</td>
</tr>
<tr>
<td>VA</td>
<td>3.18%</td>
<td>1.36%</td>
<td>12.52</td>
<td>2.14</td>
<td>8.67</td>
<td>24.85</td>
<td>4.12b</td>
</tr>
<tr>
<td>WA</td>
<td>1.91%</td>
<td>1.23%</td>
<td>15.61</td>
<td>3.10</td>
<td>12.49</td>
<td>29.78</td>
<td>4.12b</td>
</tr>
<tr>
<td>WV</td>
<td>4.21%</td>
<td>1.90%</td>
<td>13.61</td>
<td>1.96</td>
<td>8.81</td>
<td>28.55</td>
<td>4.13a</td>
</tr>
<tr>
<td>WI</td>
<td>2.91%</td>
<td>1.50%</td>
<td>13.82</td>
<td>2.33</td>
<td>9.80</td>
<td>28.86</td>
<td>4.13a</td>
</tr>
<tr>
<td>WY</td>
<td>1.08%</td>
<td>5.24%</td>
<td>15.07</td>
<td>0.94</td>
<td>4.63</td>
<td>38.33</td>
<td>4.13a</td>
</tr>
</tbody>
</table>

Table 4.2: Summary of Commuting Trips by State
To compare results to empirical data, trip length needs to be converted to time. Commute time, not distance, is the metric usually analyzed by the U.S. Census and its associated surveys. Assuming a 30 mile per hour commute speed would allow easy, yet rough, translation from trip length to trip time as 2 minutes per mile. A summary of the average travel time across the United States up until 2009 is shown in Figure 4.14.

The 2009 national average commute time was 25.1 minutes, and the average commute trip length was 12.09 miles (U.S. Department of Transportation, 2009). The average commute length across all states in the simulation is 14.09 miles. Without accounting for the circuitry of roads versus straight line distances, this average is higher than survey results. However, median trip lengths of simulated commutes for all states fall within the right range, so the distribution is likely distorted by outliers, which should qualify for modes of transportation other than a Taxis. In addition, assuming the uniform commute speed of 30 miles per hour, the simulated national median commute length of 12.72 miles, approximately 25.44 minutes duration, corresponds well to the actual time data. The U.S. Census estimates that nearly 10% of trips take over 60 minutes. While this metric depends on a confluence of factors (like traffic and transportation networks), one would expect the length of the commute to be nearly 30 miles. In most, if not all, commute distance distributions the 25 - 30 mile trip length qualifies the 90-th percentile which is consistent with the empirical data.

However, state level numbers only tell part of the story. There is significant variation in commuting characteristics within a state. Figure 4.15 shows the median commute length for the 63 counties.
of New York, which has dense urban centers as well as vast rural areas. For reference, the state-wide median trip length is 7.94 miles. There are counties with generally short commutes that are near urban centers, such as New York City, Albany, and Buffalo. Commute lengths are computed using an inverse distance squared model, so there is an expected skew towards short trips where attraction is high. The longest median commutes are for Suffolk and Nassau counties, located on Long Island. This is due to the stark contrast in density of employment opportunities between Eastern Long Island and New York City, which forces Long Island workers into New York City. Along with the Hudson Valley counties in Southern New York, these counties are naturally drawn to employment opportunities in New York City. Empirically, these counties are known to have the longest commutes in the country, which is partially attributable due to the volume and subsequent traffic congestion of commuting trips to NYC. The distribution of commute lengths within New York State is consistent with what the model should produce and the empirical reality of commuting in the state.

![Median Commute Trip Length by County in New York](image)

Figure 4.15: Median Trip Length by County in New York State

The spatial distribution of commuting trips is an essential component of an accurate disaggregated model. To narrow in on the veracity and nature of the geographic spread of commutes, Addison County, located in Vermont, is analyzed in detail. Addison County has a population of 36,821 resi-
Of those, 17,747 are workers commuting on a typical day. The Journey-to-Work Census is the main data input behind the flow of residents from county to county. The Journey-to-Work flows for Addison County detail the movements of 19,172 workers. The discrepancy between the two figures is two-fold: (1) The Journey-to-Work Census uses estimates that have a margin of error and (2) the number of workers in a county is dependent on the contrived traveler type distributions, which are subject to estimates for unemployment, those taking a sick day, etc. In addition, one should not expect the Journey-to-Work aggregate numbers to match exactly the true movements on any given day. However, one should expect that the proportion of workers going from Addison County to each of the 41 possible counties of work in the simulation matches the Journey-to-Work. Figure 4.16 shows the geographical flow of Addison County working residents to their county of work, accompanied by Table 4.3 which details the numerical flow of workers. The spatial distribution is as expected: most workers who live in Addison County also work there, and the number of workers declines as distance radiates from their home county. Interestingly, there are surrounding pockets of metropolitan areas that pull workers from Addison County, although in very small numbers. There are travelers who, for work purposes, that go to New York City and District of Columbia, as well as Detroit and Minneapolis.
Table 4.3: Addison County Worker Flows

<table>
<thead>
<tr>
<th>Work County</th>
<th>Estimate</th>
<th>Simulated</th>
<th>Work County</th>
<th>Estimate</th>
<th>Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addison (VT)</td>
<td>13,411</td>
<td>13,076</td>
<td>Kings (NY)</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Chittenden (VT)</td>
<td>4,088</td>
<td>3,323</td>
<td>Westchester (NY)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Rutland (VT)</td>
<td>862</td>
<td>714</td>
<td>Wayne (MI)</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Washington (VT)</td>
<td>282</td>
<td>218</td>
<td>Saratoga (NY)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Windsor (VT)</td>
<td>129</td>
<td>98</td>
<td>Philadelphia (PA)</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Franklin (VT)</td>
<td>42</td>
<td>38</td>
<td>Caledonia (VT)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Orange (VT)</td>
<td>39</td>
<td>33</td>
<td>Middlesex (CT)</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Essex (NY)</td>
<td>33</td>
<td>38</td>
<td>Suffolk (MA)</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Grafton (NH)</td>
<td>26</td>
<td>22</td>
<td>Worcester (MA)</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Albany (NY)</td>
<td>26</td>
<td>19</td>
<td>Cumberland (PA)</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>International (NA)</td>
<td>26</td>
<td>20</td>
<td>Richmond City (VA)</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Rensselaer (NY)</td>
<td>23</td>
<td>18</td>
<td>Fairfield (CT)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>D.C. (D.C.)</td>
<td>21</td>
<td>16</td>
<td>Norfolk (MA)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Middlesex (MA)</td>
<td>15</td>
<td>9</td>
<td>Dakota (MN)</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Washington (NY)</td>
<td>13</td>
<td>9</td>
<td>Merrimack (NH)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Franklin (NY)</td>
<td>11</td>
<td>9</td>
<td>Gloucester (NJ)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Strafford (NY)</td>
<td>10</td>
<td>10</td>
<td>Bennington (VT)</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Passaic (NJ)</td>
<td>10</td>
<td>9</td>
<td>Lamoille (VT)</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Orleans (VT)</td>
<td>10</td>
<td>8</td>
<td>Rockingham (NY)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Grand Isle (VT)</td>
<td>9</td>
<td>7</td>
<td>Sullivan (NH)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Baltimore City (MD)</td>
<td>8</td>
<td>7</td>
<td>Non-Workers (NA)</td>
<td>19,074</td>
<td></td>
</tr>
</tbody>
</table>

While accurate county-to-county travel is a fundamental aspect of the model, the microscopic characterization of commutes is equally important. Figure 4.17 shows the actual linear movements of the workers within three random Census blocks within Addison County, which illustrates two important characteristics of this model. First, it provides a visualization of the aggregation of residents to the centroid of their home Census block. All trips of residents within the block radiate from an exact point of latitude and longitude. Secondly, it demonstrates the specificity of the spatial representation of the commutes.

### 4.3 Home to School Trips

The second most important anchor of daily travel is school. This model assigns all residents who are within the ages of 6 and 24 years an exact school. Because of the geographical restrictions in assigning schools, the model yields cumulative distributions of trip lengths by states that follow a distinctly different profile than work commutes. As one might expect of typical daily trips to school, the significant majority of trips are less than 10 miles. The 2001 National Household Transportation Survey stated that in 2001 approximately 25% of students traveled less than one mile, 18% traveled
between one and two miles, and 57% traveled greater than two miles. The distributions of the simulated student trips are very closely in line with these general brackets. (U.S. Department of Transportation, 2008) In the following graphs and summary of student trip lengths for all 51 states or state equivalents, it is clear the model produced results that match existing data, and follow intuition about typical student travel behavior.

---

3Data on student travel are not available in the 2010 Census
Figure 4.18: Cumulative Distribution of Trip Length For School Trips by State (AL - CO)

(a) Alabama, Alaska, Arizona

(b) Arkansas, California, Colorado

Figure 4.19: Cumulative Distribution of Trip Length For School Trips by State (CT - HI)

(a) Connecticut, D.C., Delaware

(b) Florida, Georgia, Hawaii

Figure 4.20: Cumulative Distribution of Trip Length For School Trips by State (ID - KY)

(a) Idaho, Illinois, Indiana

(b) Iowa, Kansas, Kentucky
Figure 4.21: Cumulative Distribution of Trip Length For School Trips by State (LA - MN)

(a) Louisiana, Maine, Maryland

(b) Massachusetts, Michigan, Minnesota

Figure 4.22: Cumulative Distribution of Trip Length For School Trips by State (MS - NH)

(a) Mississippi, Missouri, Montana

(b) Nebraska, Nevada, New Hampshire

Figure 4.23: Cumulative Distribution of Trip Length For School Trips by State (NJ- OH)

(a) New Jersey, New Mexico, New York

(b) North Carolina, North Dakota, Ohio
Figure 4.24: Cumulative Distribution of Trip Length For School Trips by State (OK - SD)

(a) Oklahoma, Oregon, Pennsylvania

(b) Rhode Island, South Carolina, South Dakota

Figure 4.25: Cumulative Distribution of Trip Length For School Trips by State (TN - WA)

(a) Tennessee, Texas, Utah

(b) Vermont, Virginia, Washington

Figure 4.26: Cumulative Distribution of Trip Length For School Trips by State (WV - WY)

(a) West Virginia, Wisconsin, Wyoming
<table>
<thead>
<tr>
<th>State</th>
<th>% &lt;1mi</th>
<th>% &gt;30mi.</th>
<th>Mean Length</th>
<th>10%</th>
<th>Median Length</th>
<th>90%</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>12.02%</td>
<td>13.18%</td>
<td>12.04</td>
<td>0.85</td>
<td>6.58</td>
<td>36.05</td>
<td>4.18a</td>
</tr>
<tr>
<td>AK</td>
<td>22.20%</td>
<td>5.10%</td>
<td>7.16</td>
<td>0.45</td>
<td>2.95</td>
<td>16.31</td>
<td>4.18a</td>
</tr>
<tr>
<td>AZ</td>
<td>20.38%</td>
<td>5.96%</td>
<td>9.72</td>
<td>0.47</td>
<td>5.36</td>
<td>23.89</td>
<td>4.18a</td>
</tr>
<tr>
<td>AR</td>
<td>17.97%</td>
<td>10.47%</td>
<td>10.24</td>
<td>0.57</td>
<td>4.61</td>
<td>30.71</td>
<td>4.18b</td>
</tr>
<tr>
<td>CA</td>
<td>24.99%</td>
<td>6.60%</td>
<td>9.28</td>
<td>0.38</td>
<td>4.46</td>
<td>24.79</td>
<td>4.18b</td>
</tr>
<tr>
<td>CO</td>
<td>25.96%</td>
<td>4.23%</td>
<td>6.39</td>
<td>0.39</td>
<td>2.96</td>
<td>14.29</td>
<td>4.18b</td>
</tr>
<tr>
<td>CT</td>
<td>20.29%</td>
<td>0.74%</td>
<td>7.04</td>
<td>0.51</td>
<td>4.09</td>
<td>18.30</td>
<td>4.19a</td>
</tr>
<tr>
<td>DC</td>
<td>30.85%</td>
<td>0.00%</td>
<td>2.45</td>
<td>0.26</td>
<td>1.99</td>
<td>5.44</td>
<td>4.19a</td>
</tr>
<tr>
<td>DE</td>
<td>19.24%</td>
<td>0.57%</td>
<td>6.19</td>
<td>0.54</td>
<td>4.16</td>
<td>15.15</td>
<td>4.19a</td>
</tr>
<tr>
<td>FL</td>
<td>16.13%</td>
<td>4.38%</td>
<td>8.25</td>
<td>0.66</td>
<td>4.75</td>
<td>19.71</td>
<td>4.19b</td>
</tr>
<tr>
<td>GA</td>
<td>14.36%</td>
<td>1.57%</td>
<td>6.76</td>
<td>0.75</td>
<td>4.12</td>
<td>16.97</td>
<td>4.19b</td>
</tr>
<tr>
<td>HI</td>
<td>27.24%</td>
<td>2.81%</td>
<td>7.34</td>
<td>0.37</td>
<td>3.11</td>
<td>18.59</td>
<td>4.19b</td>
</tr>
<tr>
<td>ID</td>
<td>23.61%</td>
<td>10.15%</td>
<td>9.33</td>
<td>0.45</td>
<td>3.36</td>
<td>30.34</td>
<td>4.20a</td>
</tr>
<tr>
<td>IL</td>
<td>23.37%</td>
<td>3.47%</td>
<td>7.71</td>
<td>0.41</td>
<td>4.17</td>
<td>20.82</td>
<td>4.20a</td>
</tr>
<tr>
<td>IN</td>
<td>19.94%</td>
<td>1.99%</td>
<td>6.62</td>
<td>0.53</td>
<td>3.72</td>
<td>17.46</td>
<td>4.20a</td>
</tr>
<tr>
<td>IA</td>
<td>27.55%</td>
<td>10.47%</td>
<td>9.53</td>
<td>0.35</td>
<td>3.62</td>
<td>30.51</td>
<td>4.20b</td>
</tr>
<tr>
<td>KS</td>
<td>26.44%</td>
<td>6.92%</td>
<td>8.00</td>
<td>0.38</td>
<td>2.99</td>
<td>25.40</td>
<td>4.20b</td>
</tr>
<tr>
<td>KY</td>
<td>15.84%</td>
<td>3.31%</td>
<td>7.70</td>
<td>0.65</td>
<td>4.50</td>
<td>20.71</td>
<td>4.20b</td>
</tr>
<tr>
<td>LA</td>
<td>16.77%</td>
<td>6.92%</td>
<td>7.85</td>
<td>0.61</td>
<td>4.09</td>
<td>21.21</td>
<td>4.21a</td>
</tr>
<tr>
<td>ME</td>
<td>20.84%</td>
<td>1.22%</td>
<td>6.21</td>
<td>0.48</td>
<td>3.69</td>
<td>15.08</td>
<td>4.21a</td>
</tr>
<tr>
<td>MD</td>
<td>14.77%</td>
<td>9.38%</td>
<td>11.32</td>
<td>0.69</td>
<td>6.46</td>
<td>29.09</td>
<td>4.21a</td>
</tr>
<tr>
<td>MA</td>
<td>21.27%</td>
<td>1.37%</td>
<td>6.98</td>
<td>0.47</td>
<td>3.83</td>
<td>19.04</td>
<td>4.21b</td>
</tr>
<tr>
<td>MI</td>
<td>17.82%</td>
<td>4.58%</td>
<td>8.54</td>
<td>0.57</td>
<td>4.85</td>
<td>21.23</td>
<td>4.21b</td>
</tr>
<tr>
<td>MN</td>
<td>20.23%</td>
<td>6.17%</td>
<td>8.42</td>
<td>0.52</td>
<td>4.05</td>
<td>21.17</td>
<td>4.21b</td>
</tr>
<tr>
<td>MS</td>
<td>13.35%</td>
<td>12.39%</td>
<td>11.93</td>
<td>0.77</td>
<td>6.42</td>
<td>32.94</td>
<td>4.22a</td>
</tr>
<tr>
<td>MO</td>
<td>20.10%</td>
<td>4.64%</td>
<td>7.94</td>
<td>0.52</td>
<td>3.93</td>
<td>22.78</td>
<td>4.22a</td>
</tr>
<tr>
<td>MT</td>
<td>25.77%</td>
<td>8.24%</td>
<td>8.34</td>
<td>0.39</td>
<td>2.62</td>
<td>24.23</td>
<td>4.22a</td>
</tr>
<tr>
<td>NE</td>
<td>27.23%</td>
<td>8.92%</td>
<td>8.54</td>
<td>0.34</td>
<td>2.99</td>
<td>26.74</td>
<td>4.22b</td>
</tr>
<tr>
<td>NV</td>
<td>21.97%</td>
<td>1.00%</td>
<td>5.42</td>
<td>0.45</td>
<td>3.72</td>
<td>11.55</td>
<td>4.22b</td>
</tr>
<tr>
<td>NH</td>
<td>17.74%</td>
<td>4.82%</td>
<td>8.76</td>
<td>0.58</td>
<td>4.90</td>
<td>22.74</td>
<td>4.22b</td>
</tr>
<tr>
<td>NJ</td>
<td>25.13%</td>
<td>2.34%</td>
<td>5.55</td>
<td>0.38</td>
<td>3.02</td>
<td>12.39</td>
<td>4.23a</td>
</tr>
<tr>
<td>NM</td>
<td>21.54%</td>
<td>12.14%</td>
<td>10.91</td>
<td>0.47</td>
<td>3.67</td>
<td>36.19</td>
<td>4.23a</td>
</tr>
<tr>
<td>NY</td>
<td>27.46%</td>
<td>1.54%</td>
<td>5.26</td>
<td>0.32</td>
<td>2.76</td>
<td>13.10</td>
<td>4.23a</td>
</tr>
<tr>
<td>NC</td>
<td>12.50%</td>
<td>4.65%</td>
<td>8.36</td>
<td>0.82</td>
<td>4.99</td>
<td>21.12</td>
<td>4.23b</td>
</tr>
<tr>
<td>ND</td>
<td>27.30%</td>
<td>16.13%</td>
<td>11.95</td>
<td>0.35</td>
<td>3.04</td>
<td>42.08</td>
<td>4.23b</td>
</tr>
<tr>
<td>OH</td>
<td>18.76%</td>
<td>3.19%</td>
<td>7.25</td>
<td>0.33</td>
<td>4.47</td>
<td>17.08</td>
<td>4.23b</td>
</tr>
<tr>
<td>OK</td>
<td>21.25%</td>
<td>9.91%</td>
<td>9.79</td>
<td>4.72</td>
<td>4.46</td>
<td>29.91</td>
<td>4.24a</td>
</tr>
<tr>
<td>OR</td>
<td>27.01%</td>
<td>3.11%</td>
<td>6.61</td>
<td>0.39</td>
<td>2.92</td>
<td>16.18</td>
<td>4.24a</td>
</tr>
<tr>
<td>PA</td>
<td>20.06%</td>
<td>1.34%</td>
<td>6.69</td>
<td>0.48</td>
<td>4.13</td>
<td>16.51</td>
<td>4.24a</td>
</tr>
<tr>
<td>RI</td>
<td>25.57%</td>
<td>2.42%</td>
<td>4.73</td>
<td>0.40</td>
<td>2.55</td>
<td>11.23</td>
<td>4.24b</td>
</tr>
<tr>
<td>SC</td>
<td>12.03%</td>
<td>7.63%</td>
<td>9.85</td>
<td>0.85</td>
<td>5.63</td>
<td>24.54</td>
<td>4.24b</td>
</tr>
<tr>
<td>SD</td>
<td>28.41%</td>
<td>11.74%</td>
<td>10.02</td>
<td>0.34</td>
<td>2.87</td>
<td>31.22</td>
<td>4.24b</td>
</tr>
<tr>
<td>TN</td>
<td>14.00%</td>
<td>5.03%</td>
<td>8.69</td>
<td>0.74</td>
<td>5.21</td>
<td>23.05</td>
<td>4.25a</td>
</tr>
<tr>
<td>TX</td>
<td>21.02%</td>
<td>6.57%</td>
<td>9.59</td>
<td>0.47</td>
<td>4.81</td>
<td>23.97</td>
<td>4.25a</td>
</tr>
<tr>
<td>UT</td>
<td>25.74%</td>
<td>6.12%</td>
<td>7.75</td>
<td>0.42</td>
<td>2.93</td>
<td>16.27</td>
<td>4.25a</td>
</tr>
<tr>
<td>VT</td>
<td>19.01%</td>
<td>4.99%</td>
<td>8.71</td>
<td>0.51</td>
<td>5.00</td>
<td>22.37</td>
<td>4.25b</td>
</tr>
<tr>
<td>VA</td>
<td>19.54%</td>
<td>2.44%</td>
<td>6.26</td>
<td>0.54</td>
<td>3.40</td>
<td>16.09</td>
<td>4.25b</td>
</tr>
<tr>
<td>WA</td>
<td>22.79%</td>
<td>3.19%</td>
<td>7.12</td>
<td>0.46</td>
<td>3.69</td>
<td>17.98</td>
<td>4.25b</td>
</tr>
<tr>
<td>WV</td>
<td>16.67%</td>
<td>7.09%</td>
<td>9.11</td>
<td>0.59</td>
<td>4.82</td>
<td>26.71</td>
<td>4.26a</td>
</tr>
<tr>
<td>WI</td>
<td>22.92%</td>
<td>8.99%</td>
<td>9.48</td>
<td>0.44</td>
<td>4.14</td>
<td>28.59</td>
<td>4.26a</td>
</tr>
<tr>
<td>WY</td>
<td>28.80%</td>
<td>17.81%</td>
<td>12.93</td>
<td>0.34</td>
<td>2.37</td>
<td>56.11</td>
<td>4.26a</td>
</tr>
</tbody>
</table>

Table 4.4: Summary of School Trips by State
A metric of accuracy for Task 3 is how well the simulated school enrollment for a given school matches the enrollment data. The model can select an appropriate school given distance and relative attraction, but the final outcome of aggregate enrollment for schools is the true measure of reality. Selecting a specific school for a given individual means drawing from a distribution. As always, there exists two options: to draw with or without replacement. To analyze the effect of this choice on the outcome of simulated school enrollment, the two scenarios were considered for New Jersey’s Mercer County public high schools. In doing so, the microscopic characteristics of school travel for a small subset of students can also be visualized. There are 14,765 students who attend 16 public high schools in Mercer County, according to the school enrollment data used. Table 4.5 shows the simulated enrollment under both scenarios compared to the actual enrollment in the 16 public high schools within Mercer County.

<table>
<thead>
<tr>
<th>High School</th>
<th>Student Enrollment</th>
<th>Percent Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With</td>
<td>Without</td>
</tr>
<tr>
<td>W. Windsor Plainsboro South</td>
<td>1154</td>
<td>1667</td>
</tr>
<tr>
<td>Hamilton East Steinert</td>
<td>1314</td>
<td>1495</td>
</tr>
<tr>
<td>Hightstown High</td>
<td>1522</td>
<td>1438</td>
</tr>
<tr>
<td>Princeton High School</td>
<td>1061</td>
<td>1405</td>
</tr>
<tr>
<td>Hamilton North Nottingham</td>
<td>1319</td>
<td>1301</td>
</tr>
<tr>
<td>Hamilton West Watson</td>
<td>1061</td>
<td>1276</td>
</tr>
<tr>
<td>Central Valley High School</td>
<td>727</td>
<td>1152</td>
</tr>
<tr>
<td>Lawrence High School</td>
<td>1084</td>
<td>1186</td>
</tr>
<tr>
<td>Ewing High School</td>
<td>1181</td>
<td>1127</td>
</tr>
<tr>
<td>Trenton Central High</td>
<td>1997</td>
<td>972</td>
</tr>
<tr>
<td>Robbinsville High School</td>
<td>710</td>
<td>890</td>
</tr>
<tr>
<td>Trenton Central HS West</td>
<td>519</td>
<td>325</td>
</tr>
<tr>
<td>Mercer Jr Sr High School</td>
<td>208</td>
<td>220</td>
</tr>
<tr>
<td>Daylight Twilight High School</td>
<td>392</td>
<td>138</td>
</tr>
<tr>
<td>MCVS Assumpunk Center</td>
<td>80</td>
<td>106</td>
</tr>
<tr>
<td>MCVS Sypek Center</td>
<td>72</td>
<td>53</td>
</tr>
<tr>
<td><strong>Mercer County Total</strong></td>
<td><strong>14,401</strong></td>
<td><strong>14,751</strong></td>
</tr>
</tbody>
</table>

Table 4.5: Mercer County Public High School Enrollment Analysis

The scenario with replacement is a relaxation of the school enrollment constraint that there is a finite number of seats at a particular school. This assumption yields an expected result: schools in high density areas where population is concentrated, such as Trenton, N.J., have simulated school enrollments that exceed their actual enrollment. Because this is a zero sum game, this comes at

---

4In the “without replacement” solution, the constraint is never allowed to be a hard cap; the distribution always maintains a non-zero weight that is exponentially decreasing towards zero. This is to discount the order in which students are simulated, and to prevent a scenario in which a student who lives right next to a particular high school is not allowed to attend it because its enrollment has been maxed out already.
the expense of other high schools in Mercer County. In both situations, the full county enrollment is within 3% of the total actual enrollment, however when using replacement, the individual school enrollment is incredibly inconsistent. When not using replacement, the school enrollments in the simulation are very close to the actual enrollment data: the average absolute error between the simulated school enrollment to the actual school enrollment is 2.75%.

Figure 4.27: School Trips for Princeton High School, Central Valley High School, Hamilton East Steinart, and Trenton Central

Figure 4.27 shows a spray chart of the filaments of travel for students of four high schools within the county. It is important to note that the line representations of travel do not represent the density of travel. For Princeton High School, while there is a large geographical spread of students relative to Trenton Central High School, the large majority of travel is within a short distance from the school. Figure 4.28 supports this distinction by presenting a cumulative distribution of trip length that is characteristic of the national simulated trend for school trips. The black markers on the map represent the other public high schools in Mercer County for reference. Without restricting public school trips to particular school districts or zones, this model permits the possibility of students, who may live close to one school, attending one further away. The Gravity Model uses distance to
discourage this attraction, but it is always statistically possible. For reference, the distance between Princeton High School and Central Valley High School is approximately nine miles.

![Cumulative Distribution of Trip Length for Home-School Trips](image)

Figure 4.28: Cumulative Distribution of Trip Length for Trips to Mercer County Public High Schools

School enrollment, as with work commuting, is an opportunity to realistically simulate a predictable, important trip for a large swathe of the population. This model achieves a reasonably correct distribution of trip distances that mimics expected student travel behavior in length. All the while, the model has the ability to honor school enrollment data and accurately distribute trip volume spatially. This is crucially important when considering applications of aTaxis. Locations such as school and workplaces are natural points of aggregation of potential passengers. When passengers aggregate in common points of origin, the rates of potential ride-sharing significantly increase. Ensuring the right volume of potential passengers at a particular location is essential for capitalizing on ride-sharing and in evaluating the efficacy of an aTaxi system.

### 4.4 All Daily Trips

Task 4 and Task 5 have the responsibility of filling in the rest of the daily trip tour, after school and work have been introduced. National averages show that residents in the U.S. take somewhere between three and four trips daily; the number has fluctuated over time. With 300 million residents within the U.S. in 2010, Task 5 completes the data set, which already has all school and commuting trips, by adding all the other trips. Table 4.6 shows the total number of trips taken by the residents
of each state. There is a total of 1,009,322,835 personal trips nationally on this typical workday. The synthesized result is 8% lower than the target total of 1.1 billion, which comes from the national 2009 average of 3.7 trips per person. The simulated result computes to 3.29 trips per person.

<table>
<thead>
<tr>
<th>State</th>
<th>Trip Count</th>
<th>State</th>
<th>Trip Count</th>
<th>State</th>
<th>Trip Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>15,825,280</td>
<td>Kentucky</td>
<td>14,349,637</td>
<td>North Dakota</td>
<td>2,196,072</td>
</tr>
<tr>
<td>Alaska</td>
<td>2,397,128</td>
<td>Louisiana</td>
<td>8,148,094</td>
<td>Ohio</td>
<td>38,087,870</td>
</tr>
<tr>
<td>Arizona</td>
<td>20,903,632</td>
<td>Maine</td>
<td>4,429,255</td>
<td>Oklahoma</td>
<td>12,262,687</td>
</tr>
<tr>
<td>Arkansas</td>
<td>9,554,065</td>
<td>Maryland</td>
<td>19,316,346</td>
<td>Oregon</td>
<td>12,730,747</td>
</tr>
<tr>
<td>California</td>
<td>123,852,078</td>
<td>Massachusetts</td>
<td>21,866,204</td>
<td>Pennsylvania</td>
<td>41,709,485</td>
</tr>
<tr>
<td>Colorado</td>
<td>16,839,860</td>
<td>Michigan</td>
<td>32,833,666</td>
<td>Rhode Island</td>
<td>3,489,284</td>
</tr>
<tr>
<td>Connecticut</td>
<td>11,850,814</td>
<td>Minnesota</td>
<td>11,167,667</td>
<td>South Carolina</td>
<td>15,262,810</td>
</tr>
<tr>
<td>DC</td>
<td>2,040,597</td>
<td>Mississippi</td>
<td>9,717,958</td>
<td>South Dakota</td>
<td>2,627,648</td>
</tr>
<tr>
<td>Delaware</td>
<td>2,970,506</td>
<td>Missouri</td>
<td>19,699,659</td>
<td>Tennessee</td>
<td>21,096,931</td>
</tr>
<tr>
<td>Florida</td>
<td>61,275,215</td>
<td>Montana</td>
<td>3,261,369</td>
<td>Texas</td>
<td>83,584,971</td>
</tr>
<tr>
<td>Georgia</td>
<td>32,302,424</td>
<td>Nebraska</td>
<td>5,979,671</td>
<td>Utah</td>
<td>9,047,267</td>
</tr>
<tr>
<td>Hawaii</td>
<td>4,437,926</td>
<td>Nevada</td>
<td>9,048,868</td>
<td>Vermont</td>
<td>2,104,664</td>
</tr>
<tr>
<td>Idaho</td>
<td>5,141,420</td>
<td>New Hampshire</td>
<td>4,444,770</td>
<td>Virginia</td>
<td>26,646,786</td>
</tr>
<tr>
<td>Indiana</td>
<td>21,431,504</td>
<td>New Mexico</td>
<td>6,779,308</td>
<td>West Virginia</td>
<td>6,106,765</td>
</tr>
<tr>
<td>Iowa</td>
<td>9,943,451</td>
<td>New York</td>
<td>64,529,719</td>
<td>Wisconsin</td>
<td>18,824,313</td>
</tr>
<tr>
<td>Kansas</td>
<td>9,327,336</td>
<td>North Carolina</td>
<td>31,638,807</td>
<td>Wyoming</td>
<td>1,870,191</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,009,322,835</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.6: Total Number of Trips Taken by State Residents

The underestimation is largely due to the diversity of trips tours considered and their relative distributions within the Traveler Type categories. First and foremost, 21 patterns is not reflective of the range of sequencing that exists empirically. Particularly, the model underestimates the number of Other trips taken by the simulated residents. The distributions and organization of Traveler Types has an implicit role in determining the distribution of trip tours, and the total number of trips. The model did not give enough trips to Traveler Type 6 - those who do not go to work but are of working age. This population is not insignificant. Future iterations should augment the distribution of their activity patterns to reflect a higher expected number of daily trips. This will increase the total number of trips within the system. Beyond this particular issue, the distributions used are contrived and are made to be reasonably reflective, however future iterations must take more care to enhance the granularity of Traveler Type categorizations, their possible activity patterns, and their respective trip tour distributions. Because analysis at the state and national level is incredibly cumbersome given the sheer volume of trips, the composition and nature of all trips, especially the Other trips introduced in Task 5, are the subject of further analysis within the forthcoming case
4.5 Time Distribution of Trips

The temporal distribution of trips is built to resemble the empirical distribution of trips by start time in a given day in Figure 4.14 in Section 3.6. The majority of trip tours allowed by the synthesizer begin with trips to work or to school at normal ‘rush hour’ peak times from 7AM to 9AM. Recall, the start times of those trips is computed based on an industry standard for either work or school. This means that all constructions workers go to the work at the same time, all retail workers go to work at the same time, etc. Consequently, most daily trip tours are initiated at the same time of day. In addition, because of the limited diversity of trip patterns - and the ordering of trips - most trips in the day are back-loaded and come after a work day or school day. A histogram of all trips by start time for Bexar County, Texas (which includes San Antonio) shows exactly what one should expect from this contrived scenario.

![Histogram of Trips by Start Time](image)

**Figure 4.29: Distribution of Trips by Start Time in Bexar County, TX**

There is a massive peak of trip start times in the 8 o’clock hour (for commutes to work that begins at 9AM). In addition, the afternoon rush out of work is similarly condensed, and most of the ‘Other’ trips come all at once after departing work. The failure of this scheme is due to the false aggregation of workers within an industry to set arrival and departure times, as well as the limited range of trip sequences, which forces more trips towards the end of the day. To show the flexibility
and possible alternatives of temporal assignment, two different scenarios are considered. Alongside it is a modified distribution that uses two different techniques for distributing trips in time and overcoming this false aggregation.

(a) Revised Arrival/Departure Window  
(b) Random Start and End Work Times

Figure 4.30: Revised Distribution of Trips by Start Time for Bexar County, TX

Figure 4.30a revises the arrival and departure time window from 10 minutes before to 20 minutes before and the expected arrival time is 10 minutes before the bell time. This allows for workers to report to work and to depart work in a more diffuse time span. Figure 4.30b uses the same arrival and departure windows but, instead of using a static bell time for an industry, it draws a random bell time from a normal distribution with a mean of the industry bell time parameter and a 15% variance. The conceptual motivation behind this is to allow for variations within industries, i.e. workers go to construction site A at 7:15AM and workers go to construction site B at 8:00AM.

For the first revised scenario, Figure 4.30a is not at all different from the original methodology, except that arrival time is shifted earlier and the departure time later, with limited variation in the peak rush hours. However, the second revised scenario produced significantly different results which resemble the empirical distribution far better. The revisions possible within Task 6 show both the room for improvement in the modeling as well as the flexibility in the model to generate significantly more realistic results. The greatest hindrance to the temporal distributions is rooted in the limited range of patterns which forces lots of ‘Other’ trips to come late in the day which makes these histograms skew left.

5See Section 2.6
6Note: the y-axis scales are different in Figure 4.30b, but due to the large relative difference to the original distribution, using the same scale would have masked the particularities of the revised distribution

59
Chapter 5

Case Studies of Simulated Trips

Assessing the data set of approximately 1 billion trips on whole is more a question of working with big data set rather than analyzing transportation behavior. To analyze the final output of the synthesizer of all the personal trips taken by American residents on a typical workday, three different localities, of different types, are chosen for focused and narrow case studies. Manhattan County, New York is one of the five boroughs of New York City and is an incredibly dense metropolitan area that not only has a large population but also draws a large number of commuters from the tri-state area of New York, New Jersey, and Connecticut. It is the archetypal urban scenario for transportation demand modeling, and an area with vexing transportation infrastructure problems. Second to be considered is Manhattan’s theoretical opposite: Fargo, North Dakota, located in Cass County, is a small environment that is approximately 1/16th the population of Manhattan. Lastly, Peoria, Illinois situates itself right in the middle of the spectrum as a prototypical middle-America suburban area. It is the hope that a more in-depth analysis of these three areas will provide insight into the workings of the model as it has been applied to all of the United States.

For each of the three case studies, ‘oTrip’ files for each county are created. An ‘oTrip’ file for a county is a file of all the trips that originate in that county, regardless of where the trip taker is from. Previously, all trip files are organized by residence. In order to analyze the travel in a particular region, though, it is necessary to find all the trips that originate there. To assemble an ‘oTrip’ file, the U.S. trip files, organized by traveler residence, are scanned and regrouped. In this way, the trips made by someone living in New Jersey but working and traveling within Manhattan are identified and subject to analysis. The following three case studies are on the respective ‘oTrip’ files of Manhattan, Fargo, and Peoria.
5.1 Manhattan, New York

New York County is coextensive with the entire island of Manhattan (the Manhattan Borough of NYC) and has a simulated population of 1,585,873. The number of simulated oTrips, that is all of the trips originating daily from Manhattan, is 8,085,055. That total translates to over five daily trips per resident, but of course Manhattan has a huge commuter population from the other boroughs and neighboring counties. There are 3,010,666 unique travelers within New York County on the simulated average workday. The U.S. Census estimates that 1.6 million workers commute into Manhattan daily. Combined with the simulated resident population, the total ‘daily’ population of the simulated Manhattan is almost exactly what one might expect in reality.

<table>
<thead>
<tr>
<th></th>
<th>All oTrips</th>
<th>To Work</th>
<th>To School</th>
<th>To Home</th>
<th>To Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Trips</td>
<td>8,085,055</td>
<td>1,159,680</td>
<td>271,450</td>
<td>3,668,502</td>
<td>2,985,423</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>14.34%</td>
<td>3.36%</td>
<td>45.37%</td>
<td>36.93%</td>
<td></td>
</tr>
<tr>
<td>Passenger Miles</td>
<td>52,759,156</td>
<td>7,135,699</td>
<td>729,444</td>
<td>26,283,530</td>
<td>18,610,483</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>13.53%</td>
<td>1.38%</td>
<td>49.82%</td>
<td>35.27%</td>
<td></td>
</tr>
<tr>
<td>Mean Trip Length (mi)</td>
<td>6.53</td>
<td>2.68</td>
<td>7.164</td>
<td>4.206</td>
<td>6.23</td>
</tr>
<tr>
<td>Median Trip Length (mi)</td>
<td>3.31</td>
<td>2.92</td>
<td>1.68</td>
<td>4.21</td>
<td>2.72</td>
</tr>
<tr>
<td>% &gt; than 60 mi.</td>
<td>0.91%</td>
<td>1.01%</td>
<td>0.00%</td>
<td>0.76%</td>
<td>0.9%</td>
</tr>
</tbody>
</table>

Table 5.1: oTrip Statistics for Manhattan, NY

There is an average of 2.69 trips per day per unique traveler that originate in Manhattan and an average of 21.38 daily passenger miles. While 2.69 trips per unique traveler seems low, it is actually consistent with the dynamics of Manhattan and its large transient population. Commuters usually just come into Manhattan for work and head home, and do their errands near home so those trips originate elsewhere. A worker who commutes into Manhattan will most often travel directly home afterwards and this is considered a ‘To Home’ oTrip for Manhattan. Consequently, the ‘To Home’ oTrips are just under 50% of all the trips. It is necessary to consider the distinction between residents and non-resident oTrips. The number of trips taken by residents of Manhattan is 5,186,666, or roughly 64% of all oTrips. This is an average of 3.2 oTrips per Manhattan resident taken in Manhattan. This result is almost exactly the simulated national average, but because these are oTrips and not just regular trips, the figure demonstrates that nearly all of Manhattan residents’ trips are confined to Manhattan. This is unsurprising given the density of attractions that is characteristic of this archetypal urban living scenario. A full breakdown of the oTrips by type is presented in Table 5.1 along with a cumulative distribution of the simulated trip lengths by type for Manhattan in Figure 5.1.
The resident to non-resident ratio of 64% is about as close to even as seen in the U.S. given the huge commuter population in Manhattan. The trip behavior is consistent with this trait in that the ‘To Home’ trips are high in volume, but as Figure 5.2 shows the length of trips taken by residents and non-residents is markedly different.

The distribution of Manhattan oTrips in time shows further distinction between the behavior of residents and non-residents.
of Manhattan residents and non-residents. The histograms of oTrips by start time in Manhattan is presented where Figure 5.3a and Figure 5.3b are for the resident and non-resident subgroups respectively. The resident subgroup initiates its trips in a manner which is consistent with what Task 6 predicts for a population throughout a given day. That the distribution of oTrips by start time resembles so closely the distribution of all trips by start time for a given population (as seen in Figure 4.29 in Section 4.5) is another piece of evidence that shows almost all of the trips taken by Manhattan residents is within Manhattan County. The non-resident distribution of oTrips by start time is exactly what you would expect from the large commuter population: there are many lunch-time trips that begin at 11AM, and the rest of the trips later in the day are either commutes out of Manhattan to home or recreational trips after work in Manhattan.

![Histogram of oTrip by Start Time](image)

(a) Residents  
(b) Non-Residents

Figure 5.3: Manhattan oTrips by Start Time

### 5.2 Fargo, North Dakota

Fargo, North Dakota is located in Cass County, but is on the border of an adjacent county, and is considered half of the Fargo-Moorhead Metropolitan Area. It is the largest city in North Dakota, with a 2010 (and simulated) population of 105,549, which represents approximately 70% of the 149,778 residents living in the county. In the average workday simulation, 524,428 daily trips, each with an average length of 14.41 miles, originate within the confines of Cass County. Taking those trips are 148,117 unique travelers. Each unique traveler takes an average of 3.54 trips and travels 51.01 miles daily.
Table 5.2: oTrip Statistics for Fargo, ND

These metrics demonstrate two facts. First, most daily travel by Cass County residents is within the county. Of all trips, 493,018 (94.10%) are taken by county residents. This is a proxy for the level of transience within the population. It shows that businesses and schools in Fargo and other localities in Cass County can service all of the transportation demand of the county. It also shows that there is not much attraction to or from neighboring North Dakota, South Dakota, and Michigan counties. Indeed, it is not a very densely populated part of the country so this is to be expected. Additionally, if the number of average trips were lower, it would either mean there are many non-resident travelers or it would mean residents need to leave the county to pursue their activities. The former option is ruled out. 3.54 average trips, however, is close to the national average, if not just above, implying the residents can pursue their activities within the county and travelers come from outside Cass County to take trips. Table 5.2 breaks down the characteristics of all oTrips taken within Cass County and the distribution of those trips by purpose (by destination node) for further analysis.

A cumulative distribution of trip length in Figure 5.4 reveals another particularity of Cass County and Fargo. Because Fargo represents the highest relative attraction for quite some distance in North Dakota, most travel goes there (but there is not a lot travel to begin with given low population). While most residents live in or near Fargo, some Cass County residents travel a significant distance to get to Fargo. Fargo is located on the edge of the county, so there are numerous oTrips that span the county. The non-uniform land-use in Cass County explains the unevenness in the trip length cumulative distribution. This is the opposite of Manhattan, which had uniform and dense land-use.

53 Peoria, Illinois

Peoria, Illinois is the name of the county, and the major city contained within it, where each has a 2010 (and simulated) population 186,494 and 118,943 respectively. With respect to Manhattan and
Fargo, Peoria is a true middle ground. Its travelers exhibit simulated travel behavior consistent with the suburban sprawl that characterizes Peoria. There are a total of 649,781 oTrips in an average workday in Peoria County. As with Cass County, North Dakota, and in contrast to Manhattan, the number of non-resident trips, 62,743, is just under 10% of the oTrips taken by Peoria County residents. There are 194,789 unique travelers who take on average 3.33 oTrips and travel an average total distance of 26.85 miles daily. This is very close to the national simulated average. The distribution of trip lengths is especially characteristic of suburbia, where the mean and median trip lengths are quite similar. That is to say: everything is close, but not that close. In Fargo, lots of trips were short, but there were also long trips. In Manhattan, most trips are quite short given the density. Table 5.3 provides further detail on the characteristics of trips by type.

<table>
<thead>
<tr>
<th>Number of Trips</th>
<th>All oTrips</th>
<th>To Work</th>
<th>To School</th>
<th>To Home</th>
<th>To Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percent of Total</td>
<td>15.47%</td>
<td>7.02%</td>
<td>41.13%</td>
<td>36.31%</td>
<td></td>
</tr>
<tr>
<td>Passenger Miles</td>
<td>4,779,165</td>
<td>969,630</td>
<td>208,541</td>
<td>2,017,773</td>
<td>1,583,220</td>
</tr>
<tr>
<td>Percent of Total</td>
<td>20.29%</td>
<td>4.36%</td>
<td>42.22%</td>
<td></td>
<td>33.13%</td>
</tr>
<tr>
<td>Mean Trip Length (mi)</td>
<td>6.53</td>
<td>9.64</td>
<td>4.57</td>
<td>7.55</td>
<td>6.71</td>
</tr>
<tr>
<td>Median Trip Length (mi)</td>
<td>4.77</td>
<td>6.89</td>
<td>3.11</td>
<td>5.07</td>
<td>4.04</td>
</tr>
<tr>
<td>% &gt; than 60 mi.</td>
<td>0.61%</td>
<td>1.17%</td>
<td>0.00%</td>
<td>0.41%</td>
<td>0.534%</td>
</tr>
</tbody>
</table>

Table 5.3: oTrip Statistics for Peoria, IL
Interestingly, Peoria exhibits a relatively long school commute, which is unique. Most telling though, of the general uniform density of attraction characteristic of suburbia, is that ‘To Work’ oTrips, ‘To Home’ oTrips, and ‘To Other’ oTrips have almost the exact same distributions. In support of this observation, Figure 5.5 shows the cumulative distribution of all oTrips by type for comparison. Again, in contrast with Fargo and Manhattan, which exhibited more dramatic levels of density on either ends of the spectrum, Peoria’s is very even-keeled.

![Cumulative Distribution of Trip Length for Peoria, IL](image)

Figure 5.5: Cumulative Distribution of Trip Length for Peoria, IL

These case studies of three very different locations throughout the U.S. demonstrate the manner in which travel behavior synthesized by this model is consistent with the characteristics of land-use and is powerful enough to exhibit the particularities of unique regions like Manhattan, Fargo, and Peoria. The results of each simulated oTrip file conveyed how people - county residents and non-residents - moved and pursued activities during an average work day. These three regions are reference points for almost all other regions within the U.S. However, the analysis done here is of just three of the 3,143 oTrip files contained within this data set and so just scratches the surface of the diverse travel behavior in the U.S. that is the subject of this simulation.
Chapter 6

Limitations, Next Steps, and Conclusion

A model of such breadth can only achieve a certain degree of depth given constraints on the quality of data inputs and the assumptions required for their application. In addition, the wide scope of the model is a vexing challenge that puts serious limitations on the memory and run-time capacity of the synthesizer itself. The objective of this thesis is to transform an existing model, shown to work reasonably well within New Jersey, to apply to every state in the U.S. and create a data set of all the trips taken by its residents. This grand task has been shown to be both feasible and of sufficient quality to provoke further research and iterations to improve the accuracy to which the trips modeled realistically represent the actual spatial and temporal distribution of real travel. This section identifies some of the weaknesses of the model and the opportunities for advancing its use in the analysis of aTaxi systems.

6.1 Limitations

6.1.1 Data Scope and Applicability

The model’s first assumption - that everyone lives at the center of a Census block - is a limitation of the varying levels of geographical data as illustrated in Section 2.7. Census tract data are applied to blocks, county data are applied to blocks, and state data are applied to blocks. All data enhancements for future iterations of the model should be focused on improving the localization of quality data.
The model makes the basic assumption that who you are - your gender, age, income, geographical residence - can explain specific daily movements. It can be used to match you to a likely place of work and patronage. In this assumption, there are significant limitations in restricted scope and clear ways in which to apply data to improve this ‘matching’ of person to activities. With respect to the synthetic population, the U.S. Census is a trove of personal data on the block level that can be used to build a more comprehensive profile of each resident. Identifying personal traits that correlate with predictable activities, such as race and gender, could be especially helpful. The model relies on distance and attraction, but understanding the more complex motivations behind personal travel demand would enhance the simulation. In addition, the make up of households could be used to better inform the dependence and correlations of trips among household members, who might carpool to school or drive each other to work.

Unfortunately, the U.S. Census is only conducted every 10 years, and populations grow and are subject to change every year. The use of the American Community Survey, which is more frequent but less ubiquitous, is necessary to adapt a 2010 synthetic population to current levels. In addition, future iterations must remain conscious of certain assumptions made about rates of unemployment and school enrollment that are used implicitly, and update them accordingly. However, there is an important distinction to be made with respect to the simulation. Creating a simulation in which a specific John Doe goes at the same time everyday to his actual place of employment, favorite grocery store, and gym is not a realistic objective. (Although this would be prized data.) The real goal is that the movements of John Doe and his other 300 million compatriot travelers match the real spatial and temporal distributions of the actual trips taken, and all the better if the trips are as specific as possible. Data on individual travel behavior – by type of trips, time of trips, etc. – can be incorporated to better characterize the population in this way.

The bulk of travel is related to employment and patronage. The veracity of these trips is entirely reliant on the data used by U.S. InfoGroup. If the New Jersey State Model is an example, rates of employment and patronage are not always entirely accurate. School enrollment data was found to be quite reliable, particularly with primary and secondary schools. However, tailoring ratios to better represent business employment and patronage will be a high value-add to the model’s accuracy.

The Traveler Type designation used is a vestige of the New Jersey State Model and is an area for improvement. Mufti’s original use of the categories was to allocate the population accurately into distinct subsets that behave in similar ways. Indeed, the proportions of Traveler Types reflect the supposed population they represent. The distinctions, however, are based on national levels of employment, enrollment, etc. This can be localized to the county level, at least. In addition, one
of the main limitations in the model is the diversity of trip patterns available to the population. Expanding both the Traveler Type categories and the trip tour patterns can allow for more realistic representations of actual travel behavior, especially within particular subgroups such as college students and different types of workers. Data exist that details the number of trips taken by gender and purpose during an average day. Introducing ‘Other’ trips by purpose can permit the use of these data. Traveler Types are currently gender neutral and there is no categorization of Other trips by purpose. The NAICS codes provide more granularity in industries that can be used to more realistically select places of patronage that can be used to match trips to purpose.

6.1.2 Run Time and Memory Limitations

The volume of computation required to simulate daily trips for over 300 million people is a burdensome task. Severe run time and memory limitations hinder the use of the model and its ability to run frequent iterations. Task 2, 3, and 5 take nearly one hour per one million residents on average. These are the three most computationally heavy steps of the model. The long clock time of the model is an issue with two main factors.

Consider Task 2: two things are required of its objective to assign every working resident to a particular place of employment. First, the module that performs Task 2 must access the list of employers for the industry of work within the county of work of that resident. The size of these files can be as big as 30MB for dense counties. Secondly, Task 2 builds a distribution from one of these lists to eventually select a specific employer. For a given residence county, the Journey-to-Work Census details anywhere from one to one hundred counties of work. When assigning one resident who lives in Mercer County, NJ, but works in Atlantic County, NJ, to an employer, the synthesizer could move immediately next to a resident who lives in Mercer County, but works in New York County, NY, and so on. A run time profile of Task 2, which is schematically identical to Tasks 3 and 5 in this respect, reveals that a large portion of time is spent reading and caching these files for any given simulation. Of course, there is a finite number of county employee files that can be cached and accessed at one instant during a simulation. As it iterates over residents one by one, if the diffusion of work counties is high - which it often is - the synthesizer must manage and re-read many employee lists against a memory constraint on the machine that limits the number of files cached. This manifests itself equally with school enrollment (although less so, due to the restricted geographical area) and again in Task 5 when it reads the same patronage files. To overcome the limitation, the only option is to introduce an intermediate step that regroups residents by their
county of work or county of school, instead of their residence.

The second run time limitation is less easily reconciled. A bulk of the run time is the cumulative time of calls to the function that calculates the distance between points of latitude and longitude. The ‘per call’ time of the function is on the order of $10^{-6}$ seconds, however the cumulative time of all the calls to this function, as a proportion of the total run time, is nearly 30% in some cases. The issue here is the sheer volume of distance calculations required. For every working resident, hundreds, and often thousands, of distance calculations are performed. For example, to match one million working residents to 100 employers requires 100 million function calls that take approximately 300 seconds. In dense areas, where population is high and the possible places of employment number in the thousands or tens of thousands, the run time quickly accumulates. This is amplified in Task 5 when for each resident - not just working ones - up to three distributions need to be created (for a maximum of three other trips). One possible solution is to sample randomly a selection from the complete Employee and Patronage lists. Instead of drawing from a distribution of 100 employers, it could instead just do 20 but this would sacrifice some integrity of the model. Unlike with the caching of data, this run time burden is entirely a result of the unavoidable volume that comes from a model with such immense scope.

Lastly, the memory requirements of the output of the synthesizer - the data set itself - pose certain challenges. On average, the final trip sequence files (which are comma separated value formatted text files for each county in state directories) are sized at 0.3 gigabytes per one million residents. In aggregate, the data set (the trip sequences) is approximately 90 gigabytes for the entire nation. Intermediate steps produce output along the way of comparable size. It is not the storage of the files that poses a limitation, but the handling of the data in such bulk and the use of it in any meaningful and efficient way. The data set is designed with unique identifiers that locate residents geographically, and with attributes that allow all of the trips taken within the system to be linked to a particular resident, and organized by time of day, origin, destination, or type. The utilization of these components of the data set can make it truly functional. With data at such scale, a SQL or real database must be created that implements these identifiers and allows for quick and fast data access and use. This should be the first of many steps to more profoundly analyze the output and to reinforce the data’s applicability to its overall mission.
6.1.3 Independence Among States

The issue of caching data belies a significant modeling limitation of the national trip synthesizer. The simulation is done piecemeal where each state is simulated independently. This engenders discontinuities in interstate travel by tacitly ignoring interstate travel with respect to rates of employment, patronage, and school enrollment. To illustrate, imagine a factory on the border between Kentucky and Ohio that is just inside Ohio’s border. It has 1000 employees. When assigning workers to workplaces in Ohio, the factory is a candidate for work place selection, and its attractiveness is a function of its 1000 employees. As Ohio is simulated, employees are assigned to the factory without replacement. When Ohio is ‘built’, there is no way for the Kentucky iteration of the synthesizer to know the number of employees already assigned to that factory. In fact, that factory’s employment rate begins afresh at 1000. This is exaggerated on state borders, but the problem exists everywhere as journeys to work span large geographical areas. The same situation is true of patronage and school enrollment. Simulating the states in parallel would require an immense allocation of RAM for the synthesizer, which is somewhat infeasible. A more appropriate compromise is to simulate neighbor states in parallel, or groups of states that have a lot of interchange, for example New York, New Jersey, and Connecticut.

6.2 Next Steps

Aside from the implied next steps to overcome limitations, there are transformations that will prime the data set for further serious research into novel transportation systems. The fourth step of the traditional four step transportation demand model, mentioned in Section 1.2, is assigning the trips taken to the existing or contrived transportation network. The goal of this synthesized data is to assist in the analysis and implementation of a system of autonomous taxi stands through the United States. The New Jersey State Model, which hypothesized a similar network within the state, translated all of the simulated trips taken into trips between aTaxi stands. To do this, New Jersey was pixelated into a rectangular grid of small square blocks with an area of 0.25 miles. The goals of applying a national data set to an aTaxi network is the same, but a national application poses unique challenges.
6.2.1 Network Assignment

The assumption that enables the pixelization of New Jersey is that, relatively, it can be considered a flat surface. Hence, the grid can be made up of perfectly square blocks with no distortion because of the spherical surface of the earth. The transformation of trips between latitude and longitude only requires a simple linear transformation in this case. The national model which enables trips that span multiple states and many miles does not have this luxury. The simple rectangular grid must be modified to properly assign all of the trips to a network of aTaxi stands if one is to consider a national model.

The most generally used projection for these methods is the Winkel tripel Projection, which minimizes distortion of area, distance, and direction in projections. One can transform all of the trips into (x,y) coordinates with respect to this orientation. A simple version using this projection for a national grid can take an origin, (0, 0), located at the intersection of 40 degrees North and 97 degrees West, which uses a common standard parallel of the U.S. as well as a reasonable meridian. However with an origin located somewhere in Oklahoma, aTaxi blocks far away on the East and West coasts will not have the same area as intended. In addition to these inherent difficulties of distortion, national travel by way of aTaxi in excess of a few hundred miles does not seem like the best use case. A more local approach would allow for easier replication of the New Jersey State Grid model, and more accurate analysis of aTaxi systems as used locally, but it would fail to accurately consider long distance trips.

6.2.2 Multi-Modal Transportation

The New Jersey State Model, through its multiple iterations, introduced multi-modal transportation on existing train networks. This did more than just increase the accuracy of the modeled travel behavior; it benefited the efficacy of the proposed aTaxi system. When large commuter trains arrive at a station, potential passengers are aggregated at a single point in space and time. This is a valuable opportunity for ride-sharing. The foundation for long-distance and international travel has been created through matching qualified long-distance travelers to close airports; however this represents an incredibly small proportion of the population. In addition, there is no scheduling of the flights yet simulated, which is an obvious feature of real air travel that plays a role in mode choice. Interstate rail networks, such as Amtrak, are a good first step. Improving air travel and implementing existing rail networks on a national scale is a difficult task, but if done state-by-state, it can advance the power of the model by great lengths.
6.2.3 Non-Resident Travel Demand

This simulation, for simplicity, only considers the legal residents of the U.S. It ignores a significant group of international workers and travelers who are within the U.S. and traveling on a given day (and U.S. residents returning from international travel). To increase the scope of the model, future iterations might consider international travel to airports and the trips that disseminate from there to final destinations like hotels, tourist attractions, and business centers.

6.3 Conclusion

The results of the first national trip demand synthesizer show its power and feasibility, and should stimulate future iterations to more accurately model transportation demand. With the framework in place, the data assembled, and the computation structure built, the 1,009,322,835 simulated single workday trips taken by 308,745,538 synthetic residents is only a start. The exactness of the demand and the profile of individuals can be enhanced greatly to improve the data set. Additional work done on transportation demand models, which balance disaggregate detail and wide geographic application, will improve the understanding of the way people move. With cutting-edge technologies in autonomous vehicle driving coming to market, the opportunity to increase personal mobility at a far cheaper cost to the consumer and society necessitates additional research of individual travel motivations and patterns. A failure to bring the study of the implementation of novel transportation systems up to speed with the technological possibilities risks stifling the enormous potential of autonomous vehicles. The data set produced by this thesis enables the first, albeit basic, insight into the individual movements of U.S. residents throughout a typical workday and how a fleet of aTaxis might take them where they need to go.
Appendix A

Technical Documentation of the Synthesizer

Because this project entails a significant programming component in the execution of all of the Tasks in the method, this chapter is dedicated to the technical component of the model with brief introductions to the programs, modules, and important aspects. The computational side to this model is not insignificant. Efficient and powerful code is a necessity to such a large scale project. The technical portion of this thesis parallels the six Tasks outlined in Chapter 2. A similarly named module accompanies each Task. The major methods of each module which execute those Tasks are included in Appendix B. Some functions pertaining to data input and handling are not shown, but are available along with the source code. All code pertaining to this thesis and its analysis can be obtained at a public GitHub account: https://github.com/awyrough/2014APHWTHESIS. In addition to the source code, there are README files with notes that detail at length the more concrete technical considerations of the synthesizer. It should be noted that the only significant alterations to enable the code to function is to manage file redirection and inputs so that each module has access to the data that it needs. All other aspects of the model are self-contained and should be portable to any computer or Python environment.

A.1 Sample Execution

In order to demonstrate the execution of the synthesizer, a sample execution sequence for the state of Texas is outlined here. The execution is outlined by the command line arguments needed
to execute the modules, as well as the input files each module needs to access and the output files each creates along the way.

1. **Task 1 Command Line Execution**: python module1.py Texas  
   **Input**: TexasCensusDemographicQuery.txt; TexasCensusGroupQuartersQuery.txt; TexasFamilyQuery.txt  
   **Output**: TexasModule1NN1stRun.csv

2. **Task 2 Command Line Execution**: python module2.py Texas  
   **Input**: TexasModule1NN1stRun.csv; All National Employee/Patronage Files by County; ACS Industry Participation by Gender by County; Journey-to-Work Census; County FIPS to County Name Dictionary  
   **Output**: TexasModule2NN1stRun.csv

3. **Task 3 Command Line Execution**: python module3.py Texas  
   **Input**: TexasModule2nn1stRun.csv; All Private/Public School Enrollment by County; All Employee/Patronage Files by County; National Post-Secondary School Enrollment; County Adjacency Lists; County FIPS to County Name Dictionary  
   **Output**: TexasModule3NN1stRun.csv

4. **Task 4 Command Line Execution**: python module4.py Texas  
   **Input**: TexasModule3NN1stRun.csv; Activity Pattern Distributions  
   **Output**: TexasModule4NN1stRun.csv

5. **Task 5 Command Line Execution**: python module5.py Texas  
   **Input**: TexasModule4NN1stRun.csv; All National Employee/Patronage Files by County; County FIPS to County Name Dictionary; County Adjacency Lists  
   **Output**: COUNTYFIPS_TexasModule5NN1stRun.csv for all counties within Texas

6. **Task 6 Command Line Execution**: python module6.py Texas  
   **Input**: All TexasModule5NN1stRun.csv County Files; Schedule (Arrival, Departure, Duration) Files for NAICS Industries  
   **Output**: COUNTYFIPS_TexasModule6NN1stRun.csv for all counties within Texas

The simulation produces four intermediate state master files that hold progressively more personal data, as shown in the sample outputs in Chapter 2. Task 5 produces a directory of all of the raw trips, with no times assigned, for each county within Texas. Task 6 reads this directory and assigns the trip time attributes, producing a new directory of all the county trip files in their final form.

---

1 For example, 48001_TexasModule5NN1stRun.csv would be the file name for county 001 with Texas (state FIPS code of 48)
Appendix B

Source Code

B.1 Module 1 Source Code

```python
# Source Code

# PROJECT: United States Trip File Generation

# Author: A.P. Hill Wyrough

# Version date: 3/15/2014

# Python 3.3

# PURPOSE: Create residents within households for block in every state within the United States.

# Gives each resident unique geographical identifiers, numeric ID's, latitude/longitude of home,

# sex, age, household identifier, household type, traveler type, and income bracket and amount.

# Creates a master resident file for state input.

# INPUTS: 2010/2012 Census Data for Each State

# DEPENDENCIES: Numpy Library, Itertools Library

# NOTES: This code is adapted from Talal Mufti and Jake Gao. Mufti wrote the initial framework for his Masters Thesis in 2012 for the state of NJ. Aside from adapting the code for a more general input,

# more data, and 51 states instead of several hard coded counties, the bulk of the original changes include:

# - Selection with replacement

# - Creation of Households (see: householdhelper and build_households)

# - Data input and output

# """

# IMPORTS

# """

import csv

import random as rd

import numpy as np

# STATIC DATA DEFINITIONS

MAIN_PATH = 'MAIN PATH ON MY COMPUTER TOWARDS FILES OF USE'
```
M_PATH = "C:\Users\Hill\Desktop\Thesis\Data"

"DEFINE AGE RANGES FROM CENSUS"

ageRanges = [(0, 4), (5, 9), (10, 14), (15, 17), (18, 19), (20, 20), (21, 21), (22, 24),
(25, 29), (30, 34), (35, 39), (40, 44), (45, 49), (50, 54), (55, 59), (60, 61), (62, 64), (65, 66),
(67, 69), (70, 74), (75, 79), (80, 84), (85, 100)]

"DEFINE INCOME BRACKET RANGES"

INCOME_BRACKETS = {
    1: (0, 9999),
    2: (10000, 14999),
    3: (15000, 24999),
    4: (25000, 34999),
    5: (35000, 49999),
    6: (50000, 74999),
    7: (75000, 99999),
    8: (100000, 149999),
    9: (150000, 199999),
    10: (200000, 1000000)
}

'INDICES IN DATA FILE'

'IMPORTANT: REMEMBER range(2,5) = 2, 3, 4 - DOES NOT INCLUDE LAST INDEX'

'MEN AT EACH AGE GROUP (DEMOGRAPHIC QUERY FILE')

M_AGE_DIST = range(10, 33)

'WOMEN AT EACH AGE GROUP (DEMOGRAPHIC QUERY FILE')

F_AGE_DIST = range(34, 57)

'MEN-WOMEN RATIO (TOTAL MEN, TOTAL WOMEN'

SEX_DIST = [9.33]

'UNDER/OVER 18 YEARS OLD (FAMILY QUERY FILE')

UNDER_OVER_EIGHTEEN = range(31, 33)

'GROUP QUARTERS BY AGE BY TYPE (GROUP QUARTERS FILE')

GROUP_QUARTERS = range(6, 48)

'HUSBAND SIZE DISTRIBUTION'

HH_DIST = range(58.65)

'HUSBAND RELATIONSHIP DISTRIBUTION'

HH_REL_DIST = range(6.31)

'NON-FAMILY TO FAMILY RATIO (FAM\NOFAM[0] = # of NONFAM, FAM\NOFAM[1] = # of FAM')

FAM\NOFAM = [17, 2]

'GLOBAL DEFINITIONS OF COUNTERS AND ARRAYS'

# ALL MALE CHILDREN
mchildren = []

# ALL FEMALE CHILDREN
fchildren = []

# ALL MALE ADULTS
madults = []

# ALL FEMALE ADULTS
fadults = []

# POPULATION COUNT IN STATE DURING SIMULATION
pop = 0

# DUMMY POPULATION COUNT
poptwo = 0

# GROUP QUARTER POP COUNT
gpop = 0

hpop = 0

spop = 0

femaleadultcount = 0

maleadultcount = 0

femalechildrencount = 0

malechildrencount = 0

# FUNCTION DEFINITIONS

# DATA INPUT HELPER FUNCTIONS

def remove_b(teststring):
    if teststring[0] == 'b':
        newstr = teststring[1:100]

77
else:
    newstr = teststring
    return newstr

'EXTRACTS THE CENSUS TRACT NUMBER FROM GEOID_2 INCLUDED IN CENSUS INCOME DATA'
def convert_tract(tract):
    sample = tract[5:11]
    sample2 = tract[8:15]
    if len(sample) == 6:
        return sample2
    else:
        return sample

'CONVERTS A TRACT NUMBER AS AN INTEGER TO A STRING FOR THE BASIS OF COMPARISON'
def tract_to_string(tract):
    return str(tract)

'RETURNS THE COLUMN INDICES USED IN THE GROUP QUERY RAW TEXT FILE'
def group_ranges():
    seq = chain(range(6), range(10, 14), range(15, 18), range(20, 24), range(25, 28), range(30, 34), range(35, 38), range(41, 45), range(46, 49), range(51, 55), range(56, 59), range(61, 65), range(66, 69))
    return seq

'RETURNS THE COLUMN INDICES USED IN THE DEMO QUERY RAW TEXT FILE'
def demo_ranges():
    seq = chain(range(8), range(12, 69))
    return seq

'DATA INPUT AND READING FROM DATA FILES'

'READ CENSUS MATRIX FOR A SPECIFIC STATE AND RETURN MATRIX/ARRAY OF ALL DATA (BLOCKS), ALONG WITH FULL TRACT EXPRESSION'
def read_census_matrix(state):
    censusFileLocation = MPATH + '\DemographicQueries\'
    fname = censusFileLocation + state + 'Query.txt'
    tra = np.loadtxt(fname, delimiter='",', skiprows=1, dtype=str, usecols=[3], converters={3: remove_b})
    uni = np.loadtxt(fname, delimiter='",', skiprows=1, dtype=str, usecols=[2], converters={2: remove_b})
    mydata = np.recfromcsv(fname, delimiter='",', usecols=demo_ranges(),
                            filling_values=np.nan, case_sensitive=True, deletechars=' ', replace_space=' ',
                            skiprows=1)
    return mydata, tra, uni

'READ GROUP QUARTERS MATRIX FOR A SPECIFIC STATE AND RETURN MATRIX/ARRAY OF ALL DATA (BLOCKS)'
def read_group_matrix(state):
    groupQuartersFileLocation = MPATH + '\GroupQuarterQueries\'
    fname = groupQuartersFileLocation + state + 'GQuery.txt'
    mydata = np.recfromcsv(fname, delimiter='",', usecols=group_ranges(),
                            filling_values=np.nan, case_sensitive=True, deletechars=' ', replace_space=' ',
                            skiprows=1)
    return mydata

'READ FAMILY RELATIVE DISTRIBUTION FOR A SPECIFIC STATE AND RETURN MATRIX OF ALL DATA (BLOCKS)'
def read_family_matrix(state):
    familyFileLocation = MPATH + '\FamilyQueries\'
    fname = familyFileLocation + state + 'FQuery.txt'
    mydata = np.recfromcsv(fname, delimiter='",', filling_values=np.nan, case_sensitive=True, deletechars=' ', replace_space=' ',
                            skiprows=1)
    unidata = np.loadtxt(fname, delimiter='",', usecols=range(2, 6), dtype=str, skiprows=1)
    return mydata, unidata

'READ INCOME MATRIX FOR A SPECIFIC STATE AND RETURN MATRIX ARRAY OF ALL DATA (TRACTS) AND RETURN CENSUS TRACT LOOK UP TABLE IN ASCENDING ORDER'
def read_income_matrix(state):
    incomeFileLocation = MPATH + '\IncomeQueries\'
    fname = incomeFileLocation + state + 'Income.csv'
    tra = np.loadtxt(fname, delimiter='",', dtype=str, skiprows=1, usecols=[1], converters={1: convert_tract})
faminco = genfromtxt(fname, delimiter='"", dtype = double, usecols=range(15,88,8), skiprows = 1)
nonfaminco = genfromtxt(fname, delimiter='"", dtype = double, usecols=range(19, 19 + (88 - 15), 8), skiprows = 1)
whereAreNaNs = isnan(faminco)
faminco[whereAreNaNs] = 0
whereAreNaNs = isnan(nonfaminco)
nonfaminco[whereAreNaNs] = 0
return faminco, nonfaminco, tra
'READ IN LATITUDES AND LONGITUDES OF EVERY BLOCK IN STATE AND RETURN ARRAY (BLOCKS)' def read_lats_lons(state):
censusFileLocation = MPATH + '\DemographicQueries\'
fname = censusFileLocation + state + ' Query.txt'
mydata = np.recfromcsv(fname, delimiter='"", filling_values=np.nan, case_sensitive=True, usecols=(8,9), deletechars=' ', replace_space=' ')
return mydata
'ASSIGNING AGE AND GENDER TO POPULATION'
'ACCEPT AGE RANGE INDEX (0, 1, 2, ... 23) AND IT RETURNS A RANDOM INTEGER BETWEEN THE ENDPOINTS OF THAT AGE RANGE. EX: get_age(1) will return a random integer between 5 and 9, inclusive'
def get_age(g = -1):
    return rd.randint(ageRanges[g][0], ageRanges[g][1]) if g != -1 else -1
'ACCEPT ARRAY OF ALL MALE RESIDENTS BY AGE AND ALL FEMALE RESIDENTS BY AGE AND BEGINS TO POPULATE each resident is populated with an age and a sex (1 for male, 0 for female)'
def createResidents(maleAgeGroup, femaleAgeGroup):
    global mchildren, madults, fchildren, fadults, poptwo
    # ITERATE OVER EACH AGE GROUP FOR MEN IN THAT BLOCK
    for i, agepop in enumerate(maleAgeGroup):
        poptwo += agepop
        for j in range(agepop):
            x = get_age(i)
            if x <= 17:
                mchildren.append([x, 1, -1])
            else:
                madults.append([x, 1, -1])
        # ITERATE OVER EACH AGE GROUP FOR WOMEN IN THAT BLOCK
        for i, agepop in enumerate(femaleAgeGroup):
            poptwo += agepop
            for j in range(agepop):
                x = get_age(i)
                if x <= 17:
                    fchildren.append([x, 0, -1])
                else:
                    fadults.append([x, 0, -1])

ACCOUNT FOR GROUP QUARTERS POPULATION'
'Using the total population of male and female adults/children, move some into group quarters to account \' for the population in each block living in group quarters. This requires separate data that has the pop \' of each quarter type in each block by age and by gender. It uses the listed indexes for quarter type'

# HOUSEHOLD TYPES
# 0: Family # 1: Non-family # 2: Correctional Facilities # 3: Juvenile Detentions # 4: Nursing homes
# 5: Other institutionalized quarters # 6: Student housing # 7: Military # 8: Other non institutionalized quarters

'Pass it row of Group Quarter data'
def get_group_quarters(r):
    global mchildren, madults, fchildren, fadults, gpop
    cfa = []; j = []; nh = []; oiq = []; sh = []; m = []; oniq = []
l = [cfa, j, nh, oiq, sh, m, oniq]
gqlist = [r[x] for x in GROUP,QUARTERS]
for i, gqsize in enumerate(gqlist):
    mod = i%7
    if i in range(0,7):
        popList = mchildren
        popRange = (14, 17)
    elif i in range(7,14):
        popList = madults
        popRange = (18, 64)
    elif i in range(14,21):
        popList = madults
        popRange = (65, 120)
    elif i in range(21, 28):
        popList = fchildren
        popRange = (14, 17)
    elif i in range(28, 35):
        popList = fadults
        popRange = (18, 64)
    elif i in range(34, 42):
        popList = fadults
        popRange = (65, 120)
    'Add them to the right group housing list if they are in the right age'
    for j in range(gqsize):
        pll = len(popList)
        if pll > 0:
            for c in range(pll):
                z = np.random.randint(0, len(popList))
                popped = popList.pop(z)
            if popped[0] >= popRange[0] and popped[0] <= popRange[1]:
                break
            else:
                popList.insert(0, popped)
                popped = -1
        break
    else:
        gpop+=1
        popped[2] = mod+2
        l[mod].append(popped)
        return l

# # # # # # # # # # # # # # # # # # # # # # # # # # # # ##

'HOUSEHOLD, INCOME ASSIGNMENT HELPER FUNCTIONS'
# # # # # # # # # # # # # # # # # # # # # # # # # # # # ##

'returns list that repeats each element index of dist "freq" times where "freq" is
the value of that element'
def expand_distribution(dist):
    vec = [[i]*int(round(x)) for i, x in enumerate(dist)]
    'way to randomly select a single item from list after expanding it. especially
convenient for binary dist (True/False, fam/non-fam, under/over 18)'
    ' note that when an index has freq zero, it is not represented at all, which causes
a problem when there are only two indeces, one with freq 0 ''
'vector to randomly select a single item from list after expanding it. especially
convenient for binary dist (True/False, fam/non-fam, under/over 18)'
def select_one(l):
    r = np.random.randint(0, len(l)) if len(l)>1 else 0
    if not l:
        return 0, 1
    else:
        val = l.pop(r)
    return val, 1

'Expand Distribution for household size, noting that household size starts at 1, not
0'
def expand_household_size(dist):
    vec = [[i]*int(round(x)) for i, x in enumerate(dist)]
    'returns the appropriate income bracket code for a given income'
def income_amount_to_code(income):
for k in INCOME_BRACKETS.keys():
    if income<=INCOME_BRACKETS[k][1] and income>=INCOME_BRACKETS[k][0] and income != 0:
        return k
    elif income == 0:
        return 0
'Match Census Tract in Demo Data to Census tract in Income Data'
'Deprecated: Unused'
def find_matching_tract(rownum, tra, traln, currentTract):
    for i in range(0, len(traln)):
        if traln[i] == tra[rownum]:
            return i
    return -1
'Match Two Unique Identifiers, Return True or False'
'Deprecated: Unused'
def create_unique_ID(family_uni_row):
    county = family_uni_row[0]
    county = county[1:10]
    tract = family_uni_row[1]
    tract = tract[1:10]
    blockg = family_uni_row[2]
    blockg = blockg[1:10]
    block = family_uni_row[3]
    block = block[1:10]
    seq = (county, tract, blockg, block)
    return seq
'Match Two Unique Identifiers, Return True or False'
'Deprecated: Unused'
def matching_value(seq1, seq2):
    if (seq1[0] == seq2[0]) and (seq1[1] == seq2[1]) and (seq1[2] == seq2[2]) and (seq1[3] == seq2[3]):
        return True
    else:
        return False
'Find Matching Row within Census Demo Data of Family Data'
'Deprecated: Unused'
def find_matching_row(census_uni_row, familyuni):
    seqCensus = create_unique_ID(census_uni_row)
    for i in range(0, len(familyuni)):
        seqFam = create_unique_ID(familyuni[i])
        if matching_value(seqCensus, seqFam) == True:
            return i
    return False

'HOUSEHOLD, INCOME, TRAVELER TYPE ASSIGNMENT'

'traveler type given age and household type'
def traveler_type(age, hht):
    if age >= 0 and age <= 5 or age > 79 or hht in [2, 3, 4, 5, 7]:
        travelType = 0
    elif age >= 5 and age <= 15:
        travelType = 1
    elif age >= 16 and age <= 18:
        if temp >= 0.99948:
            travelType = 2
        else:
            travelType = 1
    elif age >= 18 and age <= 22 or hht == 6:
        if temp <= 0.9034:
            travelType = 3
    else:
        travelType = 1
travelType=4
def get_hh_income(famIncome, nonFamIncome, hht):
    if hht:
        i = nonFamIncome
    else:
        i = famIncome
    i = expand_distribution(i)
    val, i = select_one(i)
    bracket = val + 1
    if bracket == 1:
        amount = rd.triangular(2000, 10000, 7500)
    else:
        amount = rd.uniform(INCOME_BRACKETS[bracket][0], INCOME_BRACKETS[bracket][1])
    return amount

def add_individual_income_tt(hhi, h):
    hhinctt = []
    l = 0
    for i, p in enumerate(h[0]):
        tt = traveler_type(p[0], 0)
        if tt in [5, 6]:
            hhinctt.append([tt, -1, 0])
            l+=1
        elif tt in [0, 1, 3]:
            hhinctt.append([tt, 0, 0])
        elif tt in [2, 4]:
            studentInc = rd.uniform(INCOME_BRACKETS[1][0], min(INCOME_BRACKETS[1][1], hhi))
            hhinctt.append([tt, 1, studentInc])
            hhi-=studentInc
    coeffs = []
    for i in range(1):
        coeffs.append(rd.random())
    s = sum(coeffs)
    incomes = [hhi*c/s for c in coeffs]
    for q in hhinctt:
        if q[1] == -1:
            inc = incomes.pop()
        q[1] = income_amount_to_code(inc)
        q[2] = inc
    return hhinctt

def household_helper(censusrow, famrow):
    global pop
    censusPop = censusrow[9] + censusrow[33]
    pop+=censusPop
    global spop
    'READ IN HOUSE SIZES'
    housesizes = expand_household_size([censusrow[x] for x in HH_DIST])
    rel = [famrow[x] for x in HH_REL_DIST]
    'READ IN POPULATION IN HOUSEHOLDS BY TYPE'
    housepop = [rel[2], rel[17]]
    'READ IN DISTRIBUTION OF HOUSEHOLDERS BY TYPE BY SEX'
    famholder = expand_distribution([rel[x] for x in [4, 5]])
    nfamholder = expand_distribution([rel[x] for x in [18, 21]])
    'READ IN NUMBER OF NON FAMILY HOUSEHOLDERS LIVING ALONE OR TOGETHER'
    nomalone = [rel[x] for x in [19, 20, 22, 23]]
    'READ IN FAMILY RELATIONS FOR FAMILY HOUSEHOLDS'

    def traveler_type(p, 0):
        if age>=22 and age<=64:
            if temp<=.78:
                travelType=5
            else:
                travelType=6
        else:
            travelType=6
        return travelType

    'RETURN INCOME FOR HOUSEHOLD GIVEN FAMILY INCOME, NON FAMILY INCOME, AND HOUSEHOLD TYPE'
    travelType=4
famrel = expand_distribution([rel[x] for x in range(6,17)])
htype = []
for i in range(len(famholder)):
    htype.append(0)
for i in range(len(nfamholder)):
    htype.append(1)
'ALL HOUSES FOR THAT BLOCK'
hhh = buildHouses(housesizes,housepop,famholder,nfamholder,htype,nofamalone,famrel)
return hhh

'CREATES ALL HOUSEHOLDS WITHIN BLOCK'
def buildHouses(housesizes,housepop,famholder,nfamholder,htype,nofamalone,famrel):
    numhouses = len(housesizes)
    allHouses = []
    allHouseHolders = []
    ' Population Counters'
inNonFamHousing = 0
inFamHousing = 0
'Householder Availability'
nonfamHouseHoldersAvailable = len(nfamholder)
famHouseHoldersAvailable = len(famholder)

'Initialize all HouseHolders within Census Block'
for i in range(numhouses):
    'Select Household Type From Distribution (0: family, 1: nonfamily)'
    hht, htype = select_one(htype)
    if (hht == 0):
        gender, famholder = select_one(famholder)
inFamHousing+=1
    else:
        gender, nfamholder = select_one(nfamholder)
inNonFamHousing+=1

    'Create Householder with Dummy Age of 30, Gender, HHT, and -1 (flag
    indicating assignment to house)'
    householder = [30, int(not gender), hht, -1]
    if (int(not gender) == 1 and (len(madults) > 0):
        if (len(madults) > 0):
            temp = madults.pop()
        elif (int(not gender) == 0 and (len(fadults) > 0):
            if (len(fadults) > 0):
                temp = fadults.pop()
        elif (len(fadults) == 0 and len(madults) == 0):
            'In the event of non-normally aged householders (what we have classified as children, draw from the oldest')
            'Children of the correct gender'
            if (int(not gender) == 1):
                if (len(mchildren) > 0):
                    temp = mchildren.pop(mchildren.index(max(mchildren)))
            else:
                if (len(fchildren) > 0):
                    temp = fchildren.pop(fchildren.index(max(fchildren)))
        else:
            (int(not gender) == 1 and (len(madults) == 0) and (len(mchildren) > 0):
                if (len(mchildren) > 0):
                    temp = mchildren.pop(mchildren.index(max(mchildren)))
            elif (int(not gender) == 1 and (len(fadults) == 0) and (len(fchildren) > 0):
                if (len(fchildren) > 0):
                    temp = fchildren.pop(fchildren.index(max(fchildren)))

    householder[0] = temp[0]
    allHouseHolders.append(householder)

    'Assign HouseHolder to House (by House size) for Non Family'
for hh in enumerate(allHouseHolders):
    hh = hh[1]
    'Male, Non Family HouseHold'
        if (nofamalone[0] != 0):
            housesizes.remove(1)
allHouses.append([0, hh[2], [hh]])
hh[3] = 0
nofamalone[0]−=1
nonfamHouseHoldersAvailable−=1
continue

'Female, Non Family Household'
    if (nofamalone[2] != 0):
        housesizes.remove(1)
        allHouses.append([0, hh[2], [hh]])
        hh[3] = 0
        nonfamalone[2]−=1
        nonfamHouseHoldersAvailable−=1
        continue

housesizes.sort()
housesizes = housesizes[::−1]

'Populate Non Family Houses with Non Family Householders and Create Household Object'
while((inNonFamHousing < housepop[1]) and (nonfamHouseHoldersAvailable > 0)):
    for hh in enumerate(allHouseHolders):
        hh = hh[1]
        if (nonfamHouseHoldersAvailable > 0):
                if len(housesizes) > 0:
                    size = housesizes.pop()
                else:
                    break
                    hh[3] = 0
                    allHouses.append([size−1, hh[2], [hh]])
                    nonfamHouseHoldersAvailable−=1
                    inNonFamHousing+=(size−1)
                    continue

'Populate Family Households for Family Householders and Create Household Object'
while(((inFamHousing < housepop[0]) and (famHouseHoldersAvailable > 0)):
    for hh in enumerate(allHouseHolders):
        hh = hh[1]
        if (famHouseHoldersAvailable > 0):
                if len(housesizes) > 0:
                    size = housesizes.pop()
                else:
                    break
                    hh[3] = 0
                    allHouses.append([size−1, hh[2], [hh]])
                    famHouseHoldersAvailable−=1
                    inFamHousing+=(size−1)
                    continue

'Populate Households with All Family Relations, Exhausting Family Relation Distribution'
for j, i in enumerate(famrel):
    for k, hh in enumerate(allHouses):
        if (hh[0] == 0): continue
        else:
            if ((i == 0 and (hh[0] > 0) and (hh[1] == 0)):
                if hh[2][0][1] == 0 and (len(madults) > 0):
                    hh[0]=−1
                    person = madults.pop()
                    hh[2].append([person[0], 1, hh[1], 0])
                break
                elif (hh[2][0][1] == 1) and (len(fadults) > 0):
                    hh[0]=−1
                    person = fadults.pop()
                    hh[2].append([person[0], 0, hh[1], 0])
                break
            if ((i in [1, 2, 3, 4]) and (hh[0] == 0) and (hh[1] == 0)):
                if ((len(mchildren) + len(fchildren)) > 1):
                    continue
r = np.random.randint(1, len(mchildren) + len(fchildren))
if (r < len(mchildren)):
    person = mchildren.pop()
    hh[2].append([person[0], 1, hh[1], 0])
    hh[0] -= 1
    break
else:
    person = fchildren.pop()
    hh[2].append([person[0], 0, hh[1], 0])
    hh[0] -= 1
    break
elif (len(mchildren) > 0):
    person = mchildren.pop()
    hh[2].append([person[0], 1, hh[1], 0])
    hh[0] -= 1
    break
elif (len(fchildren) > 0):
    person = fchildren.pop()
    hh[2].append([person[0], 0, hh[1], 0])
    hh[0] -= 1
    break
elif ((i in [5, 6, 7, 8, 9, 10]) and (hh[0] > 0) and (hh[1] == 0)):
    r = np.random.randint(1, len(madults) + len(fadults))
    if (r < len(madults)):
        person = madults.pop()
        hh[2].append([person[0], 1, hh[1], 0])
        hh[0] -= 1
        break
    else:
        person = famults.pop()
        hh[2].append([person[0], 0, hh[1], 0])
        hh[0] -= 1
        break
elif (len(madults) > 0):
    person = madults.pop()
    hh[2].append([person[0], 1, hh[1], 0])
    hh[0] -= 1
    break
elif (len(fadults) > 0):
    person = famults.pop()
    hh[2].append([person[0], 0, hh[1], 0])
    hh[0] -= 1
    break
for i, hh in enumerate(allHouses):
    while (hh[0] > 0):
        if ((len(madults)+len(fadults)) > 1):
            r = np.random.randint(1, len(madults) + len(fadults))
            if (r < len(madults)):
                person = madults.pop()
                hh[2].append([person[0], 1, hh[1], 0])
                hh[0] -= 1
                break
            else:
                person = famults.pop()
                hh[2].append([person[0], 0, hh[1], 0])
                hh[0] -= 1
                break
        elif (len(madults) > 0):
            person = madults.pop()
            hh[2].append([person[0], 1, hh[1], 0])
            hh[0] -= 1
            break
        elif (len(fadults) > 0):
            person = famults.pop()
            hh[2].append([person[0], 0, hh[1], 0])
            hh[0] -= 1
            break
break

elif ((len(mchildren) + len(fchildren)) > 1):
    r = np.random.randint(1, len(mchildren) + len(fchildren))
    if (r < len(mchildren)):
        person = mchildren.pop()
        hh[2].append([person[0], 1, hh[1], 0])
        hh[0] -= 1
        break
    else:
        person = fchildren.pop()
        hh[2].append([person[0], 0, hh[1], 0])
        hh[0] -= 1
        break

elif (len(mchildren) > 0):
    person = mchildren.pop()
    hh[2].append([person[0], 1, hh[1], 0])
    hh[0] -= 1
    break

elif (len(fchildren) > 0):
    person = fchildren.pop()
    hh[2].append([person[0], 0, hh[1], 0])
    hh[0] -= 1
    break

elif (len(fchildren) == 0) and (len(mchildren) == 0) and (len(madults) == 0):
    break

'Fail Safe to Ensure All Population in Households are placed within house,
relaxing house size constraint for'

'Largest House in Block'

if (len(allHouses) > 0):
    while (len(madults) > 0):
        person = madults.pop()
        allHouses[len(allHouses) - 1][2].append([person[0], 1, 0, 0])
        allHouses[len(allHouses) - 1][0] -= 1
    while (len(fadults) > 0):
        person = fadults.pop()
        allHouses[len(allHouses) - 1][2].append([person[0], 0, 0, 0])
        allHouses[len(allHouses) - 1][0] -= 1
    while (len(fchildren) > 0):
        person = fchildren.pop()
        allHouses[len(allHouses) - 1][2].append([person[0], 0, 0, 0])
        allHouses[len(allHouses) - 1][0] -= 1
    del allHouseHolders
return allHouses

def person_writer(state, pW, l, gql, ll, fi, nfi, county, tract, block):
    global HH_COUNT
    global PERSON_COUNT
    global writeCount
    global HOUSE_COUNT
    for i, h in enumerate(l):
        house = [h[2]]
        if len(house) != 0:
            HH_COUNT += 1
            hhIncome = get_hh_income(fi, nfi, h[1])
            indIncome = add_individual_income_tt(hhIncome, house)
            for j, p in enumerate(house[0]):
                HOUSE_COUNT += 1
                PERSON_COUNT+= 1
                idnum = str(1000000000 + PERSON_COUNT)
                pid = str(state) + idnum[1:]
                writeCount+=1
                pW.writerow([state] + [county] + [tract] + [block] + [HH_COUNT] + [p
                [2]] + [l[0]] + [l[1]] + [pid])

    def person_writer(state, pW, l, gql, ll, fi, nfi, county, tract, block):
```python
for k, quarter in enumerate(gql):
    if len(quarter) != 0:
        HH_COUNT += 1
        for z, q in enumerate(quarter):
            PERSON_COUNT += 1
            idnum = str(1000000000 + PERSON_COUNT)
            pid = str(state) + idnum[1:]
            income = 0
            writeCount += 1
            pW.writerow([state] + [county] + [tract] + [block] + [HH_COUNT] + [q[2]] + [ll[0]] + [ll[1]] + [pid] + [q[0]] + [q[1]] + [traveler_type(q[0], q[2])] + [income] + [income])

# READ IN STATIC DATA

def read_states():
    """ Read in State names, abbreviations, and state codes """
    censusFileLocation = MPATH + '\ListofStates.txt'
    fname = censusFileLocation + 'ListofStates.txt'
    mydata = np.recfromcsv(fname, delimiter=';', case_sensitive=True, deletechars=' ', replace_space=' ')
    return mydata

def write_headers(pW):
    """ Write Headers to NN File """
    pW.writerow(['Residence State'] + ['County Code'] + ['Tract Code'] + ['Block Code'] + ['HH ID'] + ['HH TYPE'] + ['Latitude'] + ['Longitude'] + ['HH COUNT'] + ['Person ID Number'] + ['Age'] + ['Sex'] + ['Traveler Type'] + ['Income Bracket'] + ['Income Amount'])

# EXECUTIVE SCRIPT

def executive(teststate, teststateabbrev):
    runTester = True
    global HH_COUNT
    global HOUSE_COUNT
    global PERSON_COUNT
    global writeCount
    startTime = datetime.now()
    print(teststateabbrev + " started at " + str(datetime.now()) + " duration: " + str(datetime.now() - startTime))
    'READ ALL INPUTS'
    censusdata, tra, uni = read_census_matrix(teststate)
    latlondata = read_lat_lons(teststate)
    groupquarterdata = read_group_matrix(teststate)
    faminco, nonfaminco, traIncome = read_income_matrix(teststate)
    familydata, unidata = read_family_matrix(teststate)
    total = len(censusdata)
    if runTester:
        statecode = censusdata[0][1]
        path = 'C:/Users/Hill/Desktop/Thesis/Data/Output/Module1/'
        HH_COUNT = 0
        HOUSE_COUNT = 0
        PERSON_COUNT = 0
        writeCount = 0
        f = open(path + str(teststate + 'Module1NN2ndRun.csv'), 'w+', encoding='utf8')
        personWriter = csv.writer(f, delimiter=' ', lineterminator = '\n')
        write_headers(personWriter)
        IncomeTractRow = 0
        IncomeTract = tra[IncomeTractRow]
        for rownum, row in enumerate(censusdata):
            if row[1] in teststate:
                HH_COUNT += 1
                for q in range(len(quarter)):
                    if q[2] == IncomeBracket:
                        IncomeAmount += q[1]\n                        IncomeTract += 1
                        personWriter.writerow([str(state)] + [str(county)] + [str(tract)] + [str(block)] + [str(HH_COUNT)] + [str(q[2])] + [str(pid)] + [str(q[0])] + [str(q[1])] + [str(traveler_type(q[0], q[2]))] + [str(income)] + [str(income)])
```
personWriter = csv.writer(f, delimiter=',', lineterminator = '\n')
'GRAB ALL RELEVANT ROWS IN DATA QUERIES'
quarterrow = groupquarterdata[rownum]
familyrow = familydata[rownum]
lallonrow = lallondata[rownum]
'FIND CENSUS TRACT IN INCOME DATA'
if incomeTract != tra[rownum]:
    IncometractRow+=1
    incomeTract = traIncome[IncometractRow]
'GRAB INCOME DATA ROWS'
family_income = faminco[IncometractRow]
on_family_income = nonfaminco[IncometractRow]
countycode = uni[rownum]
'INITIALIZATION OF POPULACE WITH SEX AND AGE'
createResidents([row[x] for x in M_AGE_DIST], [row[x] for x in F_AGE_DIST])
'ACCOUNT FOR RESIDENTS IN GROUP QUARTERS'
gqlist = get_group_quarters(quarterrow)
'BUILD HOUSEHOLDS WITHIN BLOCK'
hhh = household_helper(row, familyrow)
'WRITE OUTPUT OF BLOCK SIMULATION'
person_writer(statecode, personWriter, hhh, gqlist, lallonrow, 
family_income, non_family_income, income, countycode, tra[rownum],
censusdata[rownum][5])
del gqlist, row, quarterrow, familyrow, lallonrow, family_income, non_family_income
del personWriter
'PERIODICALLY UPDATE STATUS OF EXECUTION'
if (rownum%1000 == 0):
    print(str(100*rownum/total) + '% Done!')
    print(teststate + " took this much time: " + str(datetime.now()-startTime))
f.close()
del censusdata, tra, uni, lallondata, groupquarterdata, faminco, nonfaminco,
traIncome, familydata, unidata
print('Write Count: ' + str(writeCount))
print(teststate + " took this much time: " + str(datetime.now()-startTime))
import sys
exec('executive(sys.argv[1], sys.argv[2])')

---

B.2 Module 2 Source Code

```python
from datetime import datetime
import csv

Project: United States Trip File Generation – Module 2
Author: A.P. Hill Wyrough
version date: 3/15/2014
Python 3.3

Purpose: This is the executive function for Task 2 (Module 2) that assigns a work place to every eligible worker. It reads in a state residence file and iterates over every resident.

Dependencies: None

Notes: The structure is inspired by Mufti’s Module 2, and get_work_county() helper function is an updated version of his.

from datetime import datetime
import csv
```
import countyAdjacencyReader
import industryReader
import workPlaceHelper

'O:PATH = "C:\Users\Hill\Desktop\Thesis\Data\Output\Module1\Second Run Complete\"

'M:PATH = "C:\Users\Hill\Desktop\Thesis\Data"

'Global Variable for Journey to Work Complete Census Data'

j2w = []

'RETURN THE WORK COUNTY GIVEN RESIDENT, GENDER, AGE, HOUSEHOLD TYPE, and TRAVELER TYPE.'

def get_work_county(homefips, hht, tt):
    global j2wDist
    if tt in [0, 1, 3, 6] or hht in [2, 3, 4, 5, 7]:
        return -1
    elif tt in [2, 4]:
        return homefips
    else:
        val = countyFlowDist.select()
        if (val[0] != '0'):
            return -2
        if (int(val[1]) > 5):
            return -2
        else:
            return val[1:]}

'READ IN ASSOCIATED STATE ABBREVIATIONS WITH STATE FIPS CODES'

def read_states():
    stateFileLocation = M:PATH + '\'
    fname = stateFileLocation + 'List of States.csv'
    lines = open(fname).readlines()
    return lines

'WRITE MODULE 2 OUTPUT HEADERS'

def writeHeaders(pW):
    pW.writerow(['Residence State'] + ['County Code'] + ['Tract Code'] + ['Block Code ']
        + ['HH ID'] + ['HH TYPE'] + ['Latitude'] + ['Longitude']
        + ['Person ID Number'] + ['Age'] + ['Sex'] + ['Traveler Type']
        + ['Income Bracket'] + ['Income Amount'] + ['Work County'] + ['Work Industry ']
        + ['Employer'] + ['Work Address'] + ['Work City'] + ['Work State']
        + ['Work Zip'] + ['Work County Name'] + ['NAISC Code'] + ['NAISC Description ']
        + ['Patron:Employee'] + ['Patrons'] + ['Employees'] + ['Work Lat ']
        + ['Work Lon'])

'EXECUTIVE FUNCTION TO ASSIGN WORKERS TO A WORK COUNTY, WORK INDUSTRY, AND WORK PLACE

def executive(state):
    global j2w
global countyFlowDist
outputPath = "C:\\Users\\Hill\\Desktop\\Thesis\\Data\\Output\\Module2\\"

'Read In J2W and Employment by Income by Industry'
j2w = countyAdjacencyReader.readJ2W()
menemp, womemp, meninco, winco = industryReader.
    read_employment_income_by_industry()

'Begin Progress Reporting Initialization'
startTime = datetime.now()
print(state + " started at: " + str(startTime))

'OPEN STATE RESIDENCY FILE'
fname = O:PATH + state + 'Module1NN2ndRun.csv'
f = open(fname, 'r')

personReader = csv.reader(f, delimiter=',', )
out = open(outputPath + str(state + 'Module1NN1stRun.csv'), 'w-', encoding='utf8')

personWriter = csv.writer(out, delimiter=',', lineterminator='\n')
writeHeaders(personWriter)

#############################################################################

'ITERATE OVER ALL RESIDENTS WITHIN STATE'
count = 0; trailingFIPS = ''
workingCounties = []
workingCountyObjects = []
for row in personReader:
  'Skip First Row'
  if count == 0:
    count += 1
    continue
  'ASSIGN WORK COUNTY−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−'
    'Get County Fips Code'
    fips = row[0]+row[1]
    if len(fips) != 5:
      fips = '0' + fips
    'Track County Code Through State File'
    if (fips != trailingFIPS):
      trailingFIPS = fips
    'Reset Home County'
    workingCounties = []
    workingCountyObjects = []
    'Initialize New County J2W Distribution'
    array = countyAdjacencyReader.get_movements(trailingFIPS, j2w)
    countFlowDist = countyAdjacencyReader.j2wDist(array)
    it , vals = countFlowDist.get_items()
    'If Distribution is Exhausted, Rebuild From Scratch (not ideal, but'
    'assumptions were made to distribution of TT that are not right'
    'FAIL SAFE: SHOULD NOT HAPPEN'
    if (countFlowDist.total_workers() == 0):
      array = countyAdjacencyReader.get_movements(trailingFIPS, j2w)
      countFlowDist = countyAdjacencyReader.j2wDist(array)
      it , vals = countFlowDist.get_items()
    'Get Gender, Age, HHT, TT, Income, HomeLat, HomeLon'
    gender = int(row[10]); age = int(row[9]); hht = int(row[5]);
    tt = int(row[11]); income = float(row[13]); lat = float(row[7]); lon = float(row[8])
    workCounty = get_work_county(fips, hht, tt)
    'ASSIGN WORK INDUSTRY AND WORK PLACE−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−'
    'Check If WorkCounty Has Not Already Been Initialized, if not, Add it'
    if workCounty not in workingCounties:
      workingCounties.append(workCounty)
      newWorkCounty = workPlaceHelper.workCounty(workCounty)
      workingCountyObjects.append([workCounty, newWorkCounty])
      'Select Employer'
      for j in workingCountyObjects:
        if j[0] == workCounty:
          workIndustry, index , employer = j[1].select_industry_and_employer( lat, lon, str(workCounty),
            gender, income, menemp, womemp, meninco, wominco)
          break
    else:
      if (workCounty == -1):
        workIndustry = '-1'
        employer = ['Non-Worker'] + ['NA']+['NA']+['NA']+['NA']+['NA'] +
          ['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +
          ['NA']
      else:
        workIndustry = '-2'
        employer = ['International Destination for Work'] + ['NA']+['NA']+['']
          ['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +['NA'] +
          ['NA']
    personWriter.writerow(row + [workCounty] + [workIndustry] + [employer[0]] + [employer[1]])
    + [employer[2]] + [employer[3]] + [employer[4]] + [employer[5]]}
B.2.1 Module 2 Industry Selection

```python
import sys
import cProfile

cProfile.run("execute(sys.argv[1])")
```

**Project**: United States Trip File Generation – Module 2

**Author**: A.P. Hill Wyrough

**Version Date**: 3/23/14

**Python**: 3.3

**PURPOSE**: This set of methods and classes provides operations enabling the selection of an industry of work for a particular worker. It used by workPlaceHelper.py. It reads in the ACS Industry Participation by Sex by Median Income and prepares that dataset for operations, extracting the relevant information.

**RELIANCES**: None

**DEPENDENCIES**: None

Note: None of this code is taken from Mufti’s Module 2 Synthesizer which performs all of these tasks in an entirely different way.

```python
"""
industryReader.py

Project: United States Trip File Generation – Module 2
Author: A.P. Hill Wyrough
version date: 3/23/14
Python 3.3
"""

import math
import random
import bisect
import numpy

'C County Employment Path'
C_PATH = 'C:Users\Hill\Desktop\Thesis\Data\Employment\CountyEmployeeFiles' '
E_PATH = 'C:Users\Hill\Desktop\Thesis\Data\Employment' '
M_PATH = "C:Users\Hill\Desktop\Thesis\Data"

'Match State Code to State Abbrev'
def match_code_abbrev(states, code):
    for i, j in enumerate(states):
        splitter = j.split(',,')
        if splitter[2] == code:
            return splitter[1]

'Read in associated state abbreviations with state FIPS codes'
def read_states():
    stateFileLocation = M_PATH + '\'
    fname = stateFileLocation + 'ListofStates.csv'
    lines = open(fname).read().splitlines()
    return lines

'Reconciles reading inputs as bytes or as unicode characters'
def remove_b(teststring):
    if len(teststring) == 0:
        return teststring
    if teststring[0] == 'b':
```
newstr = teststring[1:100]
else:
    newstr = teststring
return newstr

# Read in County Employment/Patronage File and Return List of All Locations in that county

def read_county_employment(fips):
    states = read_states()
    abbrev = match_code_abbrev(states, fips[0:2])
    filepath = C_PATH + abbrev + '\' + fips[0:2] + abbrev + '_EmpPatFile.csv'
    f = open(filepath, 'r+')
    data = []
data2 = numpy.loadtxt(filepath, delimiter=',', dtype=object)
    for row in f:
        data.append(row.split(','))
    return data2

# Create Distribution of Work Industries Within A County

def read_county_industries(countydata):
    iset = []
    ind = []
    for j in countydata:
        industry = j[9][2:4]
        print(industry)
        if (industry == 'NA'): industry = '99'
        if (industry not in ind):
            ind.append(industry)
            iset.append([industry, 0])
        'Increment Weight of Industry by Number of Employees There'
        for k in iset:
            if k[0] == industry: k[1]+=int(j[12][2:].strip('"'))
    return iset

# Draw at Random An Industry From Weighted List, Return Industry Code, New Distribution, Number of Work Spots Left

def select_industry(industrydist):
    keys = []
    vals = []
    for j in industrydist:
        keys.append(j[0])
        vals.append(j[1])
    variate = random.random() * sum(vals)
cum = 0.0
    for j in industrydist:
        cum += j[1]
        if variate < cum:
            temp = int(j[1])-1
            j[1] = (temp)
            return j[0], industrydist, sum(vals)-1
    return j, industrydist, sum(vals)-1

# Returns NAISC Code Corresponding to Index of Distribution

def dist_index_to_naisc_code(index):
    indexcode = [(0, 11), (1, 21), (2, 23), (3, 31), (4, 42), (5, 44),
                (6, 48), (7, 22), (8, 51), (9, 52), (10, 53), (11, 54),
                (12, 55), (13, 56), (14, 61), (15, 62), (16, 71), (17, 72),
                (18, 81), (19, 92)]
    return str(indexcode[index][1])

# Read in and create 4 lists of employment and income by gender by industry for each county

def read_employment_income_by_industry():
    filepath = E_PATH + 'SexByIndustryByCounty_MOD.csv'
    f = open(filepath, 'r+')
    menempdata = []; womempdata = []
    menincodata = []; womincodata = []
count = 0
    for row in f:
        menemp = []; womemp = []
        meninco = []; wominco = []
if count == 0: count+=1; continue
splitter = row.split(',
'Create Men Employment By Industry, Women Employment, Men Median Income By
Industry, Women Med Income'
menemp.append(splitter[0]); menemp.append(splitter[1]); menemp.append(splitter[2])
wmemp.append(splitter[0]); wmemp.append(splitter[1]); wmemp.append(splitter[2])
meninco.append(splitter[0]); meninco.append(splitter[1]); meninco.append(splitter[2])
wominco.append(splitter[0]); woninco.append(splitter[1]); woninco.append(splitter[2])
for j in (range(29, 101, 12)):
total = float(splitter[j-2])
menemp.append(float(splitter[j]) * total / 100.0)
wmemp.append(float(splitter[j+2]) * total / 100.0)
meninco.append(float(splitter[j+6]))
wominco.append(float(splitter[j+8]))
for j in [113, 125, 137, 161, 173, 197, 209, 221, 245, 257, 281, 293, 305, 317]:
total = float(splitter[j-2])
menemp.append(float(splitter[j]) * total / 100.0)
wmemp.append(float(splitter[j+2]) * total / 100.0)
meninco.append(float(splitter[j+6]))
wominco.append(float(splitter[j+8]))
menempdata.append(menemp)
wmempdata.append(wmemp)
menincodata.append(meninco)
wominco.append(wominco)
f.close()
return menempdata, wmempdata, menincodata, woninco
def cdf(weights):
total = sum(weights)
result = []
cumsum = 0
for w in weights:
cumsum += w
result.append(cumsum / total)
return result
def get_work_industryA(workcounty, gender, income, menempdata, wmempdata, menincodata, woninco, markers):
'Non-Worker'
if workcounty == '-1':
    return -1, -1
'International Worker'
if workcounty == '-2':
    return -2, -2
'Normal In-Country Worker'
count = 0
for j in menempdata:
    if workcounty == j[1]:
        index = count
        break
count+=1
'Grab Distributions According to Gender of Worker'
if gender == 0:
    empdata = menempdata[index][3:]
    incdata = menincodata[index][3:]
else:
    empdata = menempdata[index][3:]
    incdata = menincodata[index][3:]
'Zero Out Industries If No Employers Exist Within Actual Employment Data'
count = 0
for j in markers:
    if (markers[count]):
        empdata[count] = 0.0
incdata[count] = 200000

'Create distribution'
incdata[:] = [(x - income)**2 for x in incdata]
drawList = [x / y for x, y in zip(empdata, incdata)]

'Create CDF Weights'
weights = cdf(drawList)
x=random.random()

'Draw From Distribution'
idx=bisect.bisect(weights, x)

'Get Industry Code'
industry = dist_index_to_naisc_code(idx)
return industry, idx

B.2.2 Module 2 Employer Selection

---

workPlaceHelper.py

Project: United States Trip File Generation – Module 2
Author: A.P. Hill Wyrough
version date: 3/23/14
Python 3.3

PURPOSE: This set of methods and classes provides operations enabling the selection of a work place for a worker. It reads in employment files for a particular county, creates lists of employers by industry, and provides classes for creating distributions and selecting employers.

Relies on access to Emp Pat Files For All States and All counties

DEPENDENCIES: industryReader.py; countyAdjacencyReader.py

NOTE: This is all original work, none of these methods are taken from Mufti’s Module 2, as they are performed entirely in a new fashion.

---

""" Initialize a work county and return the entire object """"
import industryReader
import countyAdjacencyReader
import random
import bisect

'Working County Object – To Hold All Emp-Pat Data For A Given County'
class workingCounty:
    'Initialize with FIPS'
    def __init__(self, fips):
        self.data = industryReader.read_county_employment(fips)
        self.county = countyAdjacencyReader.read_data(fips)
        self.county.set_lat_lon()
        self.lat, self.lon = self.county.get_lat_lon()
        self.industries = []
        self.create_industryLists()
        self.distributions = []
        self.spots = []
        self.create_industry_distributions()

    'Print County For Testing Purposes'
    def printCounty(self):
        self.county.print_county()

    'Partition Employers/Patrons into Industries'
    def create_industryLists(self):
        agr = []; mqo = []; con = []; man = []; wtr = []; rtr = []
        tra = []; uti = []; inf = []; fin = []; rer = []; pro = []
        mgt = []; adm = []; edu = []; hea = []; art = []; aco = []
        otr = []; pub = []
        for j in self.data:
if j[9][2:4] == 'NA': code = 99
else: code = int(j[9][2:4])
if (code == 11): agr.append(j)
elif (code == 21): mqo.append(j)
elif (code == 23): con.append(j)
elif (code in [31, 32, 33]): man.append(j)
elif (code in [44, 45]): rtr.append(j)
elif (code in [48, 49]): tra.append(j)
elif (code == 22): uti.append(j)
elif (code == 51): inf.append(j)
elif (code == 52): fin.append(j)
elif (code == 53): rer.append(j)
elif (code == 54): pro.append(j)
elif (code == 55): mgt.append(j)
elif (code == 56): adm.append(j)
elif (code == 61): educ.append(j)
elif (code == 62): hea.append(j)
elif (code == 71): art.append(j)
elif (code == 72): aco.append(j)
elif (code == 81): otr.append(j)
elif (code == 92): pub.append(j)
else: otr.append(j)
self.industries = [agr, mqo, con, man, rtr, tra, uti, inf, fin, rer, pro, mgt, adm, educ, hea, art, aco, otr, pub]
return

'Create Distributions For Each Industry'
def create_industry_distributions(self):
distributions = []
allSpots = []
for j in self.industries:
dist = []
spots = []
count = 0
for k in j:
dist.append([count, int(k[13][2:].strip('"')), float(k[15][2:].strip('"')), float(k[16][2:].strip('"'))])
count += 1
distensions.append(dist)
allSpots.append(spots)
self.distributions = distributions
self.spots = allSpots

'Create a Gravity Model Distribution, CDF, Then Selection Of Employer'
def create_specific_distribution(self, dist, spots, homelat, homelon):
'Get List of # of Workers, Calculate Distance From Home to all Employers'
'Calculate Pre-Normalized Weighted List (# of workers / Dij^2) for all employers j in county i'
drawList = spots_to_distances(dist, spots, homelat, homelon)
if len(dist) == 0: print('ERROR')
if sum(drawList) <= 0:
    if len(drawList) - 1 > 0:
        idx = random.randint(0, len(drawList) - 1)
    else:
        idx = 0
    return(idx)
'Calculate Normalization Factor: sum of (# of workers / Dij^2) for all employers j in county i'
weightedList = weight_my_list(drawList)
'Draw From Weighted List and Get Row Pointer of Employer'
weights = industryReader.cdf(weightedList)
x=random.random()
idx=bisect.bisect(weights, x)
'return Row Pointer/Index'
return idx

'Selection of Industry and Employer for a Particular Resident, Given Work County and Demographic Data'
def select_industry_and_employer(self, lat, lon, wC, gender, income, menemp, womemp, meninco, wominco):
makers = []
for j in (self.distributions):
    if len(j) == 0:
        markers.append(True)
    else:
        markers.append(False)
indust, index = industryReader.get_work_industryA(wC, gender, income, menemp, womemp, meninco, wominco, markers)
employer = self.select_employer(index, lat, lon, wC)
return indust, index, employer

def weight_my_list(drawList):
normFactor = sum(drawList)
weightedList = [x/normFactor for x in drawList]
return weightedList

def spots_to_distances(dist, spots, lat, lon):
    'Get List of # of Workers, Calculate Distance From Home to all Employers'
    'Calculate Pre-Normalized Weighted List (# of workers / Dij^2) for all employers j in county i'
drawList = [float(s)/(countyAdjacencyReader.distance_between_points(lat, lon, j[2], j[3])**2) for s, j in zip(spots, dist)]
return drawList

B.3 Module 3 Source Code

' ' ' module3.py
Project: United States Trip File Generation - Module 3
Author: A.P. Hill Wyrough
version date: 3/15/2014
Python 3.3

Purpose: This is the executive function for Task 3 (Module 3) that assigns a school to every resident that is of a school age, or attends college.

Dependencies: schoolCounty.py

Notes: This procedure, because of its national scale, is entirely different than Mufti's statewide designations of community or non-community schools.

' ' ' from datetime import datetime
import schoolCounty
import csv
import random
import bisect
import module3classdump

'Direction of School Data Base'

'Constants for National Enrollment in Private and Public Schools'

public_school_enrollment_elem_mid = 34637.0
public_school_enrollment_high = 14668.0
private_school_enrollment_elem_mid = 4092.0
private_school_enrollment_high = 1306.0
total = public_school_enrollment_elem_mid + public_school_enrollment_high +
        private_school_enrollment_elem_mid + private_school_enrollment_high
privotal = private_school_enrollment_elem_mid + private_school_enrollment_high
pubtotal = public_school_enrollment_elem_mid + public_school_enrollment_high

'Using National Percentages of Enrollment in Private vs. Public, Use Ratios to Scale State Enrollment in Each'
def scale_public_and_private(projHigh, projElemMid):
    # NATIONAL NUMBERS TO BE SCALLED TO STATE LEVEL NUMBERS
elemmidtotal = private_school_enrollment_elem_mid +
    public_school_enrollment_elem_mid
    # PROJECTED STATE LEVEL NUMBERS FOR ENROLLMENT IN ALL SCHOOLS
    prop1 = private_school_enrollment_elem_mid / elemmidtotal
    prop2 = public_school_enrollment_elem_mid / elemmidtotal
    projectedPrivElemMid = prop1 * projElemMid
    projectedPublElemMid = prop2 * projElemMid
    prop1 = privateSchool_enrollment_high / hightotal
    prop2 = publicSchool_enrollment_high / hightotal
    projectedPrivHigh = prop1 * projHigh
    projectedPublHigh = prop2 * projHigh
    return projectedPrivElemMid, projectedPublElemMid, projectedPrivHigh,
    projectedPublHigh

'Read State Enrollment in Schools, Scaled Using Past Data'
def read_state_enrollment(state):
    fileLocation = schoolDataBase + 'statehighelemidenrollment.csv'
    f = open(fileLocation, 'r+')
    for row in f:
        row = row.split('.
        row = [row[x].strip('"') for x in range(0,(len(row)))]
        if row[0].strip('.').strip('') == state:
            statetotalenrollment2009 = float(row[8].strip('\n').strip('"'))
            statetotalenrollment2006 = float(row[1].strip('\n').strip('"'))
            statetotalenrollment2007 = float(row[4].strip('\n').strip('"'))
            statehighenrollment2006 = float(row[3].strip('"'))
            statehighenrollment2007 = float(row[6].strip('"'))
            statehighenrollment2008 = float(row[5].strip('"'))
            prop1 = statehighenrollment2006 / statetotalenrollment2006
            prop2 = statehighenrollment2007 / statetotalenrollment2007
            projected2009high = ((prop1+prop2)/2.0) * statetotalenrollment2009
            projected2009elemmid = ((prop1+prop2)/2.0) * statetotalenrollment2009
            return projected2009high, projected2009elemmid

'Read Enrollment In State For Post–Secondary Schools by Type'
def read_post_sec_enrollment(state):
    fileLocation = schoolDataBase + 'stateenrollmentindegrees.csv'
    f = open(fileLocation, 'r+', encoding = 'utf8')
    for row in f:
        row = row.split('.
        if (row[0]) == state:
            total = row[3]
            bachelor = row[4]
            graduate = row[5]
            associates = row[6].strip('\n')
            return float(total), float(bachelor)+float(graduate), float(associates),
            float(row[2])

'Create Cumulative Distribution'
def cdf(weights):
    total = sum(weights)
    result = []
    cumsum = 0
    for w in weights:
        cumsum += w
        result.append(cumsum/total)
    return result

'Assign Student a Type of School (Private/Public) or (Elem, Mid, High, College) Based on Age/HHT/State'
def get_school_type(age, gender, hht, homecounty, homestate, privelemmidpop,
    pubelemmidpop, privhighpop, pubhighpop,
fouryear, twoyear, nondeg):

'Not A Student'
if hht in [2,3,4,5,7,8] or age<5 or age>24:
    return 'non student', 'no', pubelemmidpop, privelemmidpop, pubhighpop,
        privhighpop, fouryear, twoyear, nondeg
elif hht == 6:
    fouryear+=1
    return 'on campus college', 'four year', pubelemmidpop, privelemmidpop,
        pubhighpop, privhighpop, fouryear, twoyear, nondeg
elif hht in [0,1]:
    # 6 to 10 -> ELEMENTARY SCHOOL
    if age < 11:
        type = 'elem'
        puborpriv = random.random()
        totalPop = pubelemmidpop + privelemmidpop
        thresh = pubelemmidpop / totalPop
        if puborpriv < thresh:
            pubelemmidpop-=1
            type2 = 'public'
        else:
            privelemmidpop-=1
            type2 = 'private'
    # 11 to 13 -> MIDDLE SCHOOL
    elif age < 14:
        type = 'mid'
        puborpriv = random.random()
        totalPop = pubelemmidpop + privelemmidpop
        thresh = pubelemmidpop / totalPop
        if puborpriv < thresh:
            pubelemmidpop-=1
            type2 = 'public'
        else:
            privelemmidpop-=1
            type2 = 'private'
    # 14 - 18ish -> HIGH SCHOOL (SOME 18's IN COLLEGE)
    elif age < 19:
        split = random.random()
        'Account for 18 Year Olds Who Are In College (approx 1/3)'
        if age != 18 or split < 0.35:
            type = 'high'
        elif age == 18 and split > 0.35:
            type = 'college';
            fouryearprop = fouryear / (fouryear +twoyear + nondeg)
            split = random.random()
            if split < fouryearprop:
                type2 = 'four year'
                fouryear-=1
            else:
                type2 = 'two year'
                twoyear-=1
        else:
            type = 'high'
            if type == 'high':
                puborpriv = random.random()
                totalPop = pubhighpop + privhighpop
                thresh = pubhighpop / totalPop
                if puborpriv < thresh:
                    pubhighpop-=1
                    type2 = 'public'
                else:
                    privhighpop-=1
                    type2 = 'private'
    elif age >= 19:
        type = 'college'
        split = random.random()
        fouryearprop = fouryear/(fouryear +twoyear + nondeg)
        twoyearprop = twoyear/(fouryear +twoyear + nondeg)
        nonprop = 1.0 - fouryearprop - twoyearprop
weights = cdf([fouryearprop, twoyearprop, nonprop])
names = ['four year', 'two year', 'non deg']
idx = bisect.bisect(weights, split)
if idx == 0: fouryear = 1
elif idx == 1: twoyear = 1
else: nondeg = 1
return type, type2, pubelemmidpop, privelemmidpop, pubhighpop, privhighpop, 
fouryear, twoyear, nondeg

'WRITE MODULE 2 OUTPUT HEADERS'
def writeHeaders(pW):
pW.writerow(['Residence State'] + ['County Code'] + ['Tract Code'] + ['Block Code'] + ['HH ID'] + ['HH TYPE'] + ['Lat'] + ['Long'] + ['Person ID Number'] + ['Age'] + ['Sex'] + ['Traveler Type'] + ['Income Bracket'] + ['Income Amount'] + ['Work County'] + ['Work Industry'] + ['Employer'] + ['Work Address'] + ['Work City'] + ['Work State'] + ['Work Zip'] + ['Work County Name'] + ['NAISC Code'] + ['NAISC Description'] + ['Patron:Employee'] + ['Patrons'] + ['Employees'] + ['Work Lat'] + ['Work Lon'] + ['School Name'] + ['SchoolLat'] + ['SchoolLon'])

'ITERATE OVER ALL RESIDENTS WITHIN A STATE, ASSIGN SCHOOL IF QUALIFIED STUDENT'
def execute(state):
    outputPath = "E:\Thesis\Output\Module 3\Third Runs\"
    Module 3 Output Path
    inputPath = "D:\Thesis\Output\Module2 Output"
    'Begin Reporting'
    startTime = datetime.now()
    print(state + " started at: " + str(startTime))
    'OPEN STATE RESIDENCY FILE'
    fname = inputPath + state + 'Module2NN2ndRun.csv'
    f = open(fname, 'r')
    personReader = csv.reader(f, delimiter=' ', lineterminator='
')
    out = open(outputPath + state + 'Module3NN3rdRun.csv', 'w+', encoding='utf8')
    personWriter = csv.writer(out, delimiter=' ', lineterminator='
')
    writeHeaders(personWriter)
    retrailingFIPS = '
    countyNameData = module3classdump.read_counties()
    states = module3classdump.read_states()
    'Gather State Enrollment Data'
    statehigh, stateelemmid = read_state_enrollment(state)
    privEleMidPop, pubEleMidPop, privHighPop, pubHighPop = scale.public_and_private(statehigh, stateelemmid)
    totalcollege, bachormore, assoc, non = read_post_sec_enrollment(state)
    'RUN'
    specCount = 0
    studentCount = 0
    count = 0
    for row in personReader:
        if count == 0: count+=1; continue
        'Gather Personal Data of Resident'
        homeCounty = row[1]; homeState = row[0]; workCounty = row[14]
        age = int(row[9]); gender = int(row[10]); hht = int(row[5])
        homelat = float(row[6]); homelon = float(row[7])
        'Update County of Residence, Prepare School County Object'
        if len(homeState+homeCounty) == 4:
            newCounty = '0'+homeState+homeCounty
        else:
            newCounty = homeState+homeCounty
        if newCounty != trailingFIPS:
            retrailingFIPS = newCounty
            print(trailingFIPS)
            homecounty = schoolCounty.schoolCounty(trailingFIPS)
            homecounty = schoolCounty.schoolCounty(trailingFIPS)
B.3.1 Module 3 School Selection

```python
homecounty.assemble_neighbory_dist(homecounty.fips)

homecounty.school_county_dist()

'FAIL-SAFE: REFRESH DISTRIBUTION IF GOES TO ZERO'
if bachormore == 0: totalcollege, bachormore, assoc, non =
    read_post_sec_enrollment(state)

'Get School Type'
type1, type2, pubEleMidPop, privEleMidPop, pubHighPop, \
privHighPop, bachormore, assoc, non = get_school_type(age, gender, hht, 
homeCounty, 

homeState, privEleMidPop, pubEleMidPop,

privHighPop, pubHighPop, bachormore, 
assoc, non)

'Get School For Student'
school = homecounty.get_school_by_type(type1, type2, homelat, homelon)

#print(school)

'Gather Output From School Selected (Need to Deal With Different Formats of 
School Data)'
if school == 1 or school == 0:
    name = 'NA'
schoollat = 'NA'
schoollon = 'NA'
county = 'NA'
else:
    if len(school) == 17:
        name = school[0]
schoollat = school[15]
schoollon = school[16]
county = module3classdump.lookup_name(school[5], module3classdump. 
match_abbr_code(states, school[3]), countyNameData)
    elif len(school) == 8:
        name = school[0]
schoollat = school[4]
schoollon = school[5]
county = module3classdump.match_abbr_code(states, school[1]) + 
school[2]
    elif len(school) == 11:
        name = school[3]
county = school[2]
schoollat = school[6]
schoollon = school[7]
    else:
        print(school)

'Keep Track of Progress'
if school == 0:
    specCount+=1
elif school != 1:
    studentCount+=1
    count+=1

'Write School Output' '(School name, county, Lat, Lon, Enrollment)'
personWriter.writerow(row + [name] + [county] + [schoollat] + [schoollon])

print(state + " took this much time: "+ str(datetime.now()-startTime))

print('students '+ str(studentCount))
print('unassigned '+ str(specCount))
print('pop '+ str(count))

import cProfile
import sys
cProfile.run("exec('executive(sys.argv[1])')")
```

Project: United States Trip File Generation – Module 3
Author: A.P. Hill Wyrough

version date: 3/15/2014

Python 3.3

Purpose: This is the helper module for module3, which assigns each student a proper place of school. This module creates and designs a school County object that houses all the enrollment data for a particular county and its geographical neighbors. It provides methods to select a county of schooling, and then a particular school given that county and type of school.

Dependencies: None

Notes:

```
import countyAdjacencyReader
import csv
import math
import random
import bisect

'File Location of School Data'
schoolDataBase = "C:\Users\Hill\Desktop\Thesis\Data\Schools\School Database\"

'Create Cumulative Distribution'
def cdf(weights):
    total = sum(weights)
    result = []
    cumsum = 0
    for w in weights:
        cumsum += w
        result.append(cumsum / total)
    return result

'SchoolCounty Object: An object for housing the entire school data for a particular county, and points to its neighbors.'
class schoolCounty:
    def __init__(self, fips):
        'Initialize County Geography'
        self.fips = fips
        self.county = countyAdjacencyReader.read_data(fips)
        self.county.set_lat_lon()

        'Read Post-Secondary Schools'
        self.twoyear, self.fouryear, self.nondeg = self.read_post_sec_schools_for_county(fips)

        'Read All Public Schools'
        self.elempublic, self.midpublic, self.highpublic = self.read_public_schools(fips)

        'Read All Private Schools'
        self.elemprivate, self.midprivate, self.highprivate = self.read_private_schools(fips)

        self.totalseats = self.get_total_seats()
        self.fouryeardist = []
        self.twoyeardist = []
        self.nondist = []
        self.neighborlyschools = []
        self.options = []

        'Find and Initialize Neighbors of Home County'
        def assemble_neighborhood_dist(self, fips):
            neighborlyschools = []
            for j in self.county.neighbors:
                neighborlyschools.append(schoolCounty(j))
            self.neighborlyschools = neighborlyschools

            'Calculate the Total Enrollment of a County'
            def get_total_seats(self):
                seats = 0
                for k in self.elempublic:
                    seats += int(k[5])
                for k in self.midpublic:
                    seats += int(k[5])
                for k in self.highpublic:
                    seats += int(k[5])
                for k in self.elemprivate:
seats += int(k[7])
for k in self,midprivate:
    seats += int(k[7])
for k in self,highprivate:
    seats+= int(k[7])
return seats

'Create a Distribution of All Counties relative to Home County'
def school_county_dist(self):
    options = []
    idx = 0
    seats = 0
    for j in self.neighborlyschools:
        for k in j,elempublic:
            seats += int(k[5])
        for k in j,midpublic:
            seats += int(k[5])
        for k in j,highpublic:
            seats += int(k[5])
        for k in j,elemprivate:
            seats += int(k[7])
        for k in j,midprivate:
            seats += int(k[7])
        for k in j,highprivate:
            seats += int(k[7])
        options.append([idx, j.fips, seats, distance_between_counties(self.county.lat, self.county.lon, j.county.lat, j.county.lon)])
    seats = 0
    idx+=1
    self.options = options

'Select A County of Schooling Given Home county and types'
def select_school_county(self, type1, type2):
    if len(self.county.neighbors) == 0:
        return 'home county'
    newOptions = []
    idx = 0
    seats = 0
    for j in self.neighborlyschools:
        if type2 == 'private':
            if type1 == 'elem':
                if len(j,elemprivate) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
            elif type1 == 'mid':
                if len(j,midprivate) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
            elif type1 == 'high':
                if len(j,highprivate) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
        elif type2 == 'public':
            if type1 == 'elem':
                if len(j,elempublic) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
            elif type1 == 'mid':
                if len(j,midpublic) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
            elif type1 == 'high':
                if len(j,highpublic) == 0:
                    newOptions.append([idx, j.fips, 0, 1000])
                else:
                    newOptions.append(self.options[idx])
        else:
            newOptions.append(self.options[idx])
    idx += 1
dists = [float(j[2]) / j[3]**2 for j in newOptions]
allDistances = []
[allDistances.append(j[3]) for j in newOptions]

if type2 == 'private':
    if type1 == 'elem':
        if len(self.elemprivate) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)
    elif type1 == 'mid':
        if len(self.midprivate) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)
    elif type1 == 'high':
        if len(self.highprivate) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)

if type2 == 'public':
    if type1 == 'elem':
        if len(self.elempublic) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)
    elif type1 == 'mid':
        if len(self.midpublic) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)
    elif type1 == 'high':
        if len(self.highpublic) != 0:
            dists.append(float(self.totalseats) / (min(allDistances) * 0.75)**2)

if sum(dists) == 0:
    return 'change'

if sum(dists) != 0:
    dists = [j / sum(dists) for j in dists]
weights = cdf(dists)
split = random.random()
idx=bisect.bisect(weights, split)
if idx == len(self.options):
    return ('home county')
else:
    return (self.neighborlyschools[idx])

'For Each Post–Secondary School List, Scale The Employee Numbers to Student Enrollment'
def scale_school_employment_to_students(self, statefouryear, statetwoyear, statenodeg):
    countyfouryear, countytwoyear, countynodeg = get_scale_factor(self.fips, self.country.statecode, statefouryear, statetwoyear, statenodeg)
totalEmployment = []
    [totalEmployment.append(int(j[len(j) - 4])) for j in self.fouryear]
totalFourEmployment = sum(totalEmployment)
totalEmployment = []
    [totalEmployment.append(int(j[len(j) - 4])) for j in self.twoyear]
totalTwoEmployment = sum(totalEmployment)
totalEmployment = []
    [totalEmployment.append(int(j[len(j) - 4])) for j in self.nondeg]
totalNonEmployment = sum(totalEmployment)
count = 0
for j in self.fouryear:
    j[len(j) - 4] = int((float(j[len(j) - 4]) / totalFourEmployment) * countyfouryear)
    fouryeardist.append([count, j[len(j) - 4]])
count+=1
for j in self.twoyear:
    j[len(j) - 4] = int((float(j[len(j) - 4]) / totalTwoEmployment) * countytwoyear)
    twoyeardist.append([count, j[len(j) - 4]])
count+=1
for j in self.nondeg:
    j[len(j) - 4] = int((float(j[len(j) - 4]) / totalNonEmployment) * countynodeg)
    count+=1
103
for j in self.nondeg:
    j[len(j) - 4] = int(float(j[len(j) - 4]) / totalNonEmployment) * counynodeg
    nondist.append([count, j[len(j) - 4]])
    count+=1
self.fouryeardist = fouryeardist
self.twoyeardist = twoyeardist
self.nondist = nondist
return fouryeardist, twoyeardist, nondist

'Select an Individual School For a Student'
def get_school_by_type(self, type1, type2, homelat, homelon):
    if type1 == 'elem' or type1 == 'mid' or type1 == 'high':
        county = self.select_school_county(type1, type2)
        if county == 'change':
            return 0
        if type2 == 'public':
            if type1 == 'elem':
                idx, school = drawSchool(self.elempublic, homelat, homelon)
                if self.elempublic[idx][5] > 1: self.elempublic[idx][5] = 1
            else:
                idx, school = drawSchool(county.elempublic, homelat, homelon)
                if county.elempublic[idx][5] > 1: county.elempublic[idx][5] = 1
            elif type1 == 'mid':
                idx, school = drawSchool(self.midpublic, homelat, homelon)
                if self.midpublic[idx][5] > 1: self.midpublic[idx][5] = 1
            else:
                idx, school = drawSchool(county.midpublic, homelat, homelon)
                if county.midpublic[idx][5] > 1: county.midpublic[idx][5] = 1
            elif type1 == 'high':
                idx, school = drawSchool(self.highpublic, homelat, homelon)
                if self.highpublic[idx][5] > 1: self.highpublic[idx][5] = 1
            else:
                idx, school = drawSchool(county.highpublic, homelat, homelon)
                if county.highpublic[idx][5] > 1: county.highpublic[idx][5] = 1
            elif type2 == 'private':
                if type1 == 'elem':
                    if len(self.elemprivate) != 0:
                        idx, school = drawSchool(self.elemprivate, homelat, homelon)
                    if self.elemprivate[idx][7] > 1: self.elemprivate[idx][7] = 1
                elif len(self.elemprivate) == 0:
                    idx, school = drawSchool(self.elempublic, homelat, homelon)
                else:
                    if len(county.elemprivate) != 0:
                        idx, school = drawSchool(county.elemprivate, homelat, homelon)
                    if county.elemprivate[idx][7] > 1: county.elemprivate[idx][7] = 1
                elif type1 == 'mid':
                    if len(self.midprivate) != 0:
                        idx, school = drawSchool(self.midprivate, homelat, homelon)
                    else:
                        idx, school = drawSchool(county.midprivate, homelat, homelon)
                    if county.midprivate[idx][7] > 1: county.midprivate[idx][7] = 1
                elif type1 == 'high':
                    if len(self.highprivate) != 0:
                        idx, school = drawSchool(self.highprivate, homelat, homelon)
                    else:
                        idx, school = drawSchool(county.highprivate, homelat, homelon)
                    if county.highprivate[idx][7] > 1: county.highprivate[idx][7] = 1
                else:
                    if len(self.highprivate) != 0:
                        idx, school = drawSchool(self.highprivate, homelat, homelon)
                    else:
                        idx, school = drawSchool(county.highprivate, homelat, homelon)
                    if county.highprivate[idx][7] > 1: county.highprivate[idx][7] = 1
idx, school = drawSchool(self.midprivate, homelat, homelon);
elif len(self.midprivate) == 0:
    idx, school = drawSchool(self.midpublic, homelat, homelon)
else:
    if len(county.midprivate) == 0:
        idx, school = drawSchool(county.midpublic, homelat, homelon)
    else:
        idx, school = drawSchool(county.midprivate, homelat, homelon);
        if (county.midprivate[idx][7] > 1): county.midprivate[idx][7] = 1

elif type1 == 'high':
    if county == 'home county':
        if len(self.highprivate) != 0:
            idx, school = drawSchool(self.highprivate, homelat, homelon);
        elif len(self.highprivate) == 0:
            idx, school = drawSchool(self.highpublic, homelat, homelon)
else:
    if county.highprivate == 0:
        idx, school = drawSchool(county.highpublic, homelat, homelon)
    else:
        idx, school = drawSchool(county.highprivate, homelat, homelon)
        if (county.highprivate[idx][7] > 1): county.highprivate[idx][7] = 1

elif type1 == 'college' or type1 == 'on campus college':
    if type2 == 'four year':
        try:
            school = self.drawCollege(self.fouryeadist, self.fouryear, homelat, homelon)
        except (ZeroDivisionError, IndexError):
            for j in self.neighborlyschools:
                try:
                    school = j.drawCollege(j.fouryeadist, j.fouryear, homelat, homelon)
                    break
            except (ZeroDivisionError, IndexError):
                test = True
        elif type2 == 'two year':
            try:
                school = self.drawCollege(self.twoyeadist, self.twoyear, homelat, homelon)
            except (ZeroDivisionError, IndexError):
                for j in self.neighborlyschools:
                    try:
                        school = j.drawCollege(j.fouryeadist, j.fouryear, homelat, homelon)
                        break
                    except (ZeroDivisionError, IndexError):
                        test = True
        elif type2 == 'non deg':
            try:
                school = self.drawCollege(self.nondist, self.nondeg, homelat, homelon)
            except (ZeroDivisionError, IndexError):
                for j in self.neighborlyschools:
                    try:
                        school = j.drawCollege(j.fouryeadist, j.fouryear, homelat, homelon)
school = j.drawCollege(j.nondist, j.nondeg, homelat, homelon)

break
except (ZeroDivisionError, IndexError):
test = True

elif typel == 'non student':
    return 1
else:
    return 0
try:
    return school
except UnboundLocalError:
    return 0

\'Draw College Institution From List\'
def drawCollege(self, schoolList, schools, homelat, homelon):
    weights = []
    weights.append(float(j[1]) / (distance_between_countries(float(j[len(j)-2]), float(j[len(j)-1]), homelat, homelon)**2)) for j in schoolList]
cdf2 = cdf(weights)
split = random.random()
idx = bisect.bisect(cdf2, split)
return schools[idx]

\'Initialize Public Schools For County\'
def read_public_schools(self, fips):
    elemPublic = []
    midPublic = []
    highPublic = []
    if elem != None:
        for j in elemPublic: j[5] = int(j[5])
        for row in elemPublic: j[5] = int(j[5])
        return elemPublic, midPublic, highPublic

\'Initialize Private Schools For County\'
def read_private_schools(self, fips):
    elemPrivate = []
    midPrivate = []
    highPrivate = []
    if elem != None:
        for j in elemPrivate: j[7] = int(j[7])
        for row in elemPrivate: j[7] = int(j[7])
if row[6] == '1':
elemprivate.append(row)
    highprivate.append(row)
midprivate = highprivate
return elemprivate, highprivate, highprivate

'Initialize Post Secondary Schools for county'
def read_post_sec_schools_for_county(self, fips):
    countyAbbrev = self.county.stateabbrev
    fileLocation = schoolDataBase + 'PostSecSchoolsByCounty' + countyAbbrev + '\\' + str(fips) + '\\' + countyAbbrev + '\'
    try:
        twoyear = open(fileLocation + 'CommunityCollege.csv', 'r')
twoyearschools = csv.reader(twoyear, delimiter=',')
except IOError:
    twoyear = None
try:
    fouryear = open(fileLocation + 'University.csv', 'r')
fourschools = csv.reader(fouryear, delimiter=',')
except IOError:
    fouryear = None
try:
    nondeg = open(fileLocation + 'NonDegree.csv', 'r')
nondegschools = csv.reader(nondeg, delimiter=',')
except IOError:
    nondeg = None
allNonDegSchools = []; allTwoYearSchools = []; allFourYearSchools = []
if twoyear != None:
    for row in twoyearschools: allTwoYearSchools.append(row)
if fouryear != None:
    for row in fourschools: allFourYearSchools.append(row)
if nondeg != None:
    for row in nondegschools: allNonDegSchools.append(row)
return allTwoYearSchools, allFourYearSchools, allNonDegSchools

'SELECT (NON-SECONDARY) SCHOOL FROM LIST USING ASSEMBLED DISTRIBUTION'
def drawSchool(schoolList, homelat, homelon):
    weights = []
    if len(schoolList[0]) == 11:
        [weights.append(float(j[5]) / (distance_between_counties(j[6], j[7], homelat, homelon))**2) for j in schoolList]
    alldist = []
    [alldist.append(distance_between_counties(j[6], j[7], homelat, homelon)) for j in schoolList]
    #return schoolList[alldist.index(min(alldist))]
    else:
        [weights.append(float(j[7]) / (distance_between_counties(j[4], j[5], homelat, homelon))**2) for j in schoolList]
    cdf2 = cdf(weights)
split = random.random()
idx = bisect.bisect(cdf2, split)
return idx, schoolList[idx]

'RETURN MILES BETWEEN LATITUDE AND LONGITUDE POINTS '
def distance_between_counties(lat1, lon1, lat2, lon2):
    degrees_to_radians = math.pi/180.0
    phi1 = (90.0 - float(lat1)) * degrees_to_radians
    phi2 = (90.0 - float(lat2)) * degrees_to_radians
    theta1 = float(lon1) * degrees_to_radians
    theta2 = float(lon2) * degrees_to_radians
    cos = math.sin(phi1) * math.sin(phi2) + math.cos(math.sin(theta1) - math.cos(theta2) + math.cos(phi1) * math.cos(phi2))
    arc = math.acos(cos)
    return arc * 3963.167

'Scale State Enrollment in Types of Post–Sec Schools by County Population'
'To Obtain County Enrollment in Different Programs'
```python
def get_scale_factor(fips, state, statefouryear, statetwoyear, statenodeg):
    statecounties = []
    CPATH = 'C:\\Users\\Hill\\Desktop\\Thesis\\Data\\WorkFlow'
    fname = CPATH + '\\\allCounties.txt'
    f = open(fname, 'r+')
totalStatePop = 0.0
    countyPop = []
    weights = []
    for line in f:
        splitter = line.split(',
        if splitter[1] == state:
            if splitter[3] not in statecounties:
                statecounties.append(splitter[3])
                totalStatePop+=float(splitter[7])
                countyPop.append([splitter[3], splitter[7]])
            for j in countyPop:
                weights.append([j[0], float(j[1])/totalStatePop])
        if j[0] == fips:
            req = (weights.pop())
        countyfouryear = req[1]*statefouryear
        countytwoyear = req[1]*statetwoyear
        countynodeg = req[1]*statenodeg
    return countyfouryear, countytwoyear, countynodeg
```

**B.4  Module 4 Source Code**

```python
""
module4.py
""
Project: United States Trip File Generation
Author: A.P. Hill Wyrough
version date: 3/15/2014
Python 3.3

PURPOSE: Assign Each Resident an Activity Pattern
INPUTS: Activity Pattern Distributions
DEPENDENCIES: None
""
import csv
from datetime import datetime
import random
import bisect

"""PATH DEFINITIONS"""
rootDrive = 'E'
rootFilePath = rootDrive + ':\\Thesis\\Output\\'
inputFileNameSuffix = 'Module3NN3rdRun.csv'
outputFileNameSuffix = 'Module4NN2ndRun.csv'
dataDrive = 'E'
dataRoot = dataDrive + ':\\Thesis\\Data\\'

"""PATH DEFINITIONS"""
def readActivityPatternDistributions():
    fileLocation = dataRoot + ':\\Trip Distributions and Times\\' + '
    TripTypeDistributions.csv'
    f = open(fileLocation, 'r+')
    zero = []; one = []; two = []; three = []; four = []; five = []; six = []
    allDistributions = [zero, one, two, three, four, five, six]
    for row in f:
        splitter = row.split(',
        count = 0
        for j in splitter:
            allDistributions[count].append(float(j.strip('n'))); count+=1
```

108
f.close()
return(allDistributions)

'Create Cumulative Distribution'
def cdf(weights):
total=sum(weights); result=[]; cumsum=0
for w in weights:
cumsum+=w
result.append(cumsum/total)
return result

'Create Cumulative Distribution'
def assignActivityPattern(travelerType, allDistributions, person):
'Revise Traveler Type. In the event of no school assigned (incredibly fringe population < 0.001%)':
if person[len(person)-3] == 'NA' and (travelerType == 3 or travelerType == 4 or travelerType == 2 or travelerType == 1):
    travelerType = 6
dist = allDistributions[travelerType]
weights = cdf(dist)
split = random.random()
idx = bisect.bisect(weights, split)
return idx

'WRITE MODULE 2 OUTPUT HEADERS'
def writeHeaders(pW):
pW.writerow([ 'Residence State' ] + [ 'County Code' ] + [ 'Tract Code' ] + [ 'Block Code' ]
            + [ 'HH ID' ] + [ 'HH TYPE' ] + [ 'Latitude' ] + [ 'Longitude' ]
            + [ 'Person ID Number' ] + [ 'Age' ] + [ 'Sex' ] + [ 'Traveler Type' ] + [ 'Income Bracket' ]
            + [ 'Income Amount' ] + [ 'Work County' ] + [ 'Work Industry' ] + [ '

'Open State File'
fileLocation = rootFilePath + 'Module 3\Third Runs\' + state + inputFileNameSuffix
outputLocation = rootFilePath + 'Module 4\' + state + outputFileNameSuffix
'Begin Reporting'
startTime = datetime.now()
print(state + " started at: "+ str(startTime))

'Open State File'
fileLocation = rootFilePath + 'Module 3\Third Runs\' + state + inputFileNameSuffix
f = open(fileLocation, 'r')
personReader = csv.reader(f, delimiter=' ', quotechar='\n')
for person in personReader:
    count += 1
    if count == -1: count+=1; continue
    assign Activity Pattern from Revised Traveler Type'
    travelerType = int(person[11])
    if travelerType == 5 and person[14] == '−2':
        activityIndex = '−5'
    else:
        activityIndex = assignActivityPattern(travelerType, distributions, person )
personWriter.writerow(person + [ activityIndex ])
count += 1
B.5  Module 5 Source Code

```python
# 'findOtherTrip.py

For each Other trip in a daily trip tour, draw from industry distributions and from patronage distributions.

import classDumpModule5
import numpy
import random
import bisect

# --- PATH DEFINITIONS

rootDrive = 'E'
rootFilePath = rootDrive + '\\Thesis\\Output\\
inputFileNameSuffix = 'Module4NN2ndRun.csv'
outputFileNameSuffix = 'Module5NN1stRun.csv'

dataDrive = 'E'
dataRoot = dataDrive + '\\Thesis\\Data\\

#
def populateActivities(personalTour, person, homeCountyPatronage, state, countyNameData):
    'Find Other Trips'
    trips = []
    count = 0
    for j in personalTour.activityPattern[2]:
        if j[0] == 'O':
            trips.append(count)
            count += 1
        for j in range(0, len(trips)):
            otherTrip = (personalTour.activities[trips[j]])
            'Get County of Start of Trip'
            if otherTrip[2] == 'H':
                originCounty = personalTour.activities[0][5]
                prev = 'H'
            elif otherTrip[2] == 'W':
                originCounty = personalTour.activities[trips[j] - 1][5]
                prev = 'W'
            elif otherTrip[2] == 'O':
                originCounty = personalTour.activities[trips[j] - 1][5]
                prev = 'O'
            elif otherTrip[2] == 'S':
                originCounty = personalTour.activities[trips[j] - 1][5]
                prev = 'S'
                'Determine Work–Work Trip (because of restriction)'
                marker = False
            startLon = personalTour.activities[trips[j] - 1][6]
            startLat = personalTour.activities[trips[j] - 1][7]
```

```
name, countyName, lat, lon, indust, homeCountyPatronage = getOtherTrip(str(originCounty), startLon, startLat, marker, homeCountyPatronage, prev)

personalTour.activities[trips[j]][4] = name
personalTour.activities[trips[j]][5] = classDumpModule5.lookup_name(countyName, state, countyNameData)
personalTour.activities[trips[j]][6] = lat
personalTour.activities[trips[j]][7] = lon
personalTour.activities[trips[j]][8] = indust

return personalTour, homeCountyPatronage

def getOtherTrip(originCounty, startLon, startLat, marker, homeCountyPatronage, prev):
    count = 0
    index = -1
    if prev == 'H':
        sets = homeCountyPatronage.homeCounties
    elif prev == 'S':
        sets = homeCountyPatronage.schoolCounties
    elif prev == 'W':
        sets = homeCountyPatronage.workCounties
    else:
        sets = homeCountyPatronage.otherCounties
    for j in sets:
        if originCounty == j.FIPS:
            index = count
            break
    if index == -1:
        if prev == 'H':
            homeCountyPatronage.homeCounties.append(patronageCounty(originCounty))
            index = len(homeCountyPatronage.homeCounties) - 1
            sets = homeCountyPatronage.homeCounties
        elif prev == 'S':
            homeCountyPatronage.schoolCounties.append(patronageCounty(originCounty))
            index = len(homeCountyPatronage.schoolCounties) - 1
            sets = homeCountyPatronage.schoolCounties
        elif prev == 'W':
            homeCountyPatronage.workCounties.append(patronageCounty(originCounty))
            index = len(homeCountyPatronage.workCounties) - 1
            sets = homeCountyPatronage.workCounties
        elif prev == 'O':
            homeCountyPatronage.otherCounties.append(patronageCounty(originCounty))
            index = len(homeCountyPatronage.otherCounties) - 1
            sets = homeCountyPatronage.otherCounties

    'County Patronage List'
    countyPatronage = sets[index]

    'Select Industry of Patronage'
    industry_weights = classDumpModule5.cdf(countyPatronage.patronCounts)
    split = random.randint()
    idx = bisect.bisect(industry_weights, split)
    if countyPatronage.patronCounts[idx] > 5:
        countyPatronage.patronCounts[idx] = 1
    lists = countyPatronage.industries[idx]

    'Select Particular Place of Patronage'
    allDistances = []

    'Note: Restrictions on Geography are built into distance calculations'
    if marker == False:
        [allDistances.append(float(j[12]) / (classDumpModule5.
        distance_between_points_normal(startLon, startLat, float(j[15]), float(j[16].strip('n')))**2)) for j in lists]
    else:
        [allDistances.append(float(j[12]) / (classDumpModule5.
        distance_between_points_w2w(startLon, startLat, float(j[15]), float(j[16].strip('n')))**2)) for j in lists]

    try:
        norm = sum(allDistances)
        [j/norm for j in allDistances]
    except ZeroDivisionError:
        [j/1.0 for j in allDistances]
    if sum(allDistances) == 0:
        index = random.randint(0, len(allDistances) - 1)
else:
    weights = classDumpModule5.cdf(allDistances)
    split = random.random()
    index = bisect.bisect(weights, split)
    if int(countyPatronage.industries[idx][index][12]) > 1:
        countyPatronage.industries[idx][index][12] = int(countyPatronage.industries[idx][index][12]) - 1
    patronagePlace = lists[index]
    'Return Other Trip Information'
    name = patronagePlace[0]
    county = patronagePlace[5]
    lat = float(patronagePlace[len(patronagePlace) - 2])
    lon = float(patronagePlace[len(patronagePlace) - 1].strip('\n'))
    industri = patronagePlace[9][0:2]
    return name, county, lat, lon, industri, homeCountyPatronage

class patronageCounty:
    'Initialize with FIPS'
    def __init__(self, fips):
        self.data = read_county_employment((fips))
        self.FIPS = str(fips)
        self.industries = []
        self.patronCounts = []
        self.create_industryLists()
        self.distributions = []
        self.spots = []
    'Partition Employers/Patrons into Industries'
    def create_industryLists(self):
        agr = []; mqo = []; con = []; man = []; wtr = []; rtr = []
        tra = []; uti = []; inf = []; fin = []; rer = []; pro = []
        mgt = []; adm = []; edu = []; hea = []; art = []; aco = []
        otr = []; pub = []
        agrCount = 0; mqoCount = 0; conCount = 0; manCount = 0; wtrCount = 0; rtrCount = 0
        traCount = 0; utiCount = 0; infCount = 0; finCount = 0; rerCount = 0; proCount = 0
        mgtCount = 0; admCount = 0; eduCount = 0; heaCount = 0; artCount = 0; acoCount = 0
        otrCount = 0; pubCount = 0
        for j in self.data:
            if j[9] == 'NA': code = 99
            else: code = int(j[9][0:2])
            'Deal With Scientific Notation'
            number = (j[12])
            if len(number) == 8:
                if number[len(number) - 3] == '+':
                    number = 10000
            else:
                number = int(j[12])
                if (code == 11): agr.append(j); agrCount+=number
                elif (code == 21): mqo.append(j); mqoCount+=number
                elif (code == 23): con.append(j); conCount+=number
                elif (code in [31, 32, 33]): man.append(j); manCount+=number
                elif (code == 42): wtr.append(j); wtrCount+=number
                elif (code in [44, 45]): rtr.append(j); rtrCount+=number
                elif (code in [48, 49]): tra.append(j); traCount+=number
                elif (code == 51): uti.append(j); utiCount+=number
                elif (code == 52): fin.append(j); finCount+=number
                elif (code == 53): rer.append(j); rerCount+=number
                elif (code == 54): pro.append(j); proCount+=number
                elif (code == 55): mgt.append(j); mgtCount+=number
                elif (code == 56): adm.append(j); admCount+=number
                elif (code == 61): edu.append(j); eduCount+=number
                elif (code == 62): hea.append(j); heaCount+=number
                elif (code == 71): art.append(j); artCount+=number
                elif (code == 72): aco.append(j); acoCount+=number
                elif (code == 81): otr.append(j); otrCount+=number
                elif (code == 92): pub.append(j); pubCount+=number
                else: otr.append(j); otrCount+=number
self.industries = [agr, mqo, con, man, wtr, rtr, tra, uti, inf, fin, rer, pro, mgt, adm, edu, hea, art, acq, otr, pub]
self.patronCounts = [int(agrCount), int(mqoCount), int(conCount), int(manCount), wtrCount, rtrCount, traCount, utiCount, infCount, finCount, rerCount, proCount, mgtCount, admCount, eduCount, heaCount, int(artCount), int(acqCount), otrCount, pubCount]

return

class patronageWarehouse:
    def __init__(self):
        self.homeCounties = []
        self.workCounties = []
        self.schoolCounties = []
        self.otherCounties = []

'Read in County Employment/Patronage File and Return List of All Locations in that county'
def read_county_employment(fips):
    if len(fips) == 4:
        fips = '0' + fips
    states = classDumpModule5.read_states()
    abbrev = classDumpModule5.match_code_abbrev(states, fips[0:2])
    file_path = dataRoot + 'Employment\CountyEmployeeFiles' + abbrev + fips + 'EmpPatFile.csv'
    f = open(file_path, 'r+')
    data = []
    for row in f:
        data.append(row.split(','))
    return data

B.5.1 Mode Split for Air Travel

---PATH DEFINITIONS---

import csv
import math
import random
from datetime import datetime

rootDrive = 'E'
rootFilePath = rootDrive + '\Thesis\Output\'
inputFileNameSuffix = 'Module5NN1stRun.csv'
outputFileNameSuffix = 'Module5NN1stRun_MODAL.csv'
dataDrive = 'E'
dataRoot = dataDrive + '\Thesis\Data\'
import csv
import math
import random
from datetime import datetime
import classDumpModule5
import countyAdjacencyReader
import bisect

counties = []
stateCode = ''

'Read All Airports and Create Distribution Where Land-Area Is Attraction'
def read_airports(stateabbrev):
    global stateCode
    filepath = dataRoot + 'workFlow\airports.csv'
    f = open(filepath, 'r+')
    reader = csv.reader(f, delimiter=',')
    stateAirports = []
    weights = []
    for row in reader:
            lat = row[22].split('-')
            lon = row[24].split('-')
            realLat = float(lat[0]) + float(lat[1]) / 60.0 + float(lat[2][0:6]) / 3600.0
            realLon = -(float(lon[0]) + float(lon[1]) / 60.0 + float(lon[2][0:6])) / 3600.0
            landArea = row[35]
            if landArea == '': landArea = 10
            weights.append(float(landArea))
            stateAirports.append([row[6], row[8], row[11], landArea, str(realLat), str(realLon), classDumpModule5.lookup_name(row[8], stateCode, counties)])
    return stateAirports, weights

'Send Traveler to International Destination'
'Send Revised Trip Tour'
def send_to_international(lat, lon, person, stateAirports):
    dist = [(float(j[3]) ** 2) / (distance_between_points_normal(lat, lon, j[4], j[5]) ** 2) for j in stateAirports]
    weights = classDumpModule5.cdf(dist)
    split = random.random()
    idx = bisect.bisect(weights, split)
    airport = stateAirports[idx]

    'Create Trip Tour'
    person[9] = 'Wm.'
    person[15] = 'Wm.'
    person[16] = 'H'
    person[17] = 'N'
    person[22] = airport[5]
    person[23] = 'N'
    person[24] = 'Wm.'
    return person

'take a plane(person, lat, lon, stateAirports)'

def take_a_plane(person, lat, lon, stateAirports):
    global counties
    global states
    dist = [(float(j[3]) ** 2) / (distance_between_points_normal(lat, lon, j[4], j[5]) ** 2) for j in stateAirports]
    weights = classDumpModule5.cdf(dist)
    split = random.random()
    idx = bisect.bisect(weights, split)
    Firstairport = stateAirports[idx]
    Secondairport = find_destination_airport(person, classDumpModule5.match.code.abbrev(states, person[19][0:2]))
    'Adjust Trips: 1st - to airport, 2nd - airport to work 3rd - other trip 4th - to hotel'
    'Change Node 4 to Real Work Place'
\[
\begin{align*}
\text{person}[31] &= 'W' \\
\text{person}[32] &= 'A' \\
\text{person}[33] &= 'O' \\
\text{person}[34] &= \text{person}[18] \\
\text{person}[35] &= \text{person}[19] \\
\text{person}[36] &= \text{person}[20] \\
\text{person}[37] &= \text{person}[21] \\
\text{person}[38] &= \text{person}[22] \\
\end{align*}
\]

'Change Node 2 to First Airport'

\[
\begin{align*}
\text{person}[15] &= 'A' \\
\text{person}[16] &= 'H' \\
\text{person}[17] &= 'A' \\
\text{person}[18] &= \text{Firstairport}[2] \\
\text{person}[19] &= \text{Firstairport}[6] \\
\text{person}[20] &= \text{Firstairport}[4] \\
\text{person}[21] &= \text{Firstairport}[5] \\
\text{person}[22] &= 'Airport' \\
\text{person}[9] &= 'A' \\
\end{align*}
\]

'Change Node 3 to Second Airport'

\[
\begin{align*}
\text{person}[23] &= 'A' \\
\text{person}[24] &= 'A' \\
\text{person}[25] &= 'W' \\
\text{person}[26] &= \text{Secondairport}[2] \\
\text{person}[27] &= \text{Secondairport}[6] \\
\text{person}[28] &= \text{Secondairport}[4] \\
\text{person}[29] &= \text{Secondairport}[5] \\
\text{person}[30] &= 'Airport' \\
\end{align*}
\]

'Change Node 5 to Hotel'

\[
\begin{align*}
\text{hotels} &= \text{find_hotels(}\text{person}[35]) \\
\end{align*}
\]

'Select Hotel'

\[
\begin{align*}
\text{dist} &= \left[ \frac{\text{float}(j[12])}{\text{distance_between_points_normal}(\text{person}[36], \text{person}[37], j[15], j[16])} \right]^{*2} \text{for } j \text{ in hotels} \\
\text{weights} &= \text{classDumpModule5.cdf(dist)} \\
\text{split} &= \text{random.random()} \\
\text{idx} &= \text{bisect.bisect(weights, split)} \\
\text{hotel} &= \text{hotels[idx]} \\
\end{align*}
\]

\[
\begin{align*}
\text{person}[39] &= 'O' \\
\text{person}[40] &= 'A' \\
\text{person}[41] &= 'N' \\
\text{person}[42] &= \text{hotel}[0] \\
\text{person}[43] &= \text{person}[35] \\
\text{person}[44] &= \text{hotel}[\text{len(hotel)} - 2] \\
\text{person}[45] &= \text{hotel}[\text{len(hotel)} - 1].\text{strip('\n')} \\
\text{person}[46] &= '72' \\
\end{align*}
\]

\[
\begin{align*}
\text{return } \text{person} \\
\end{align*}
\]

\[
\begin{align*}
\text{def find_destination_airport(person, destinationCounty):} \\
\text{destinationState} &= \text{destinationCounty[0:2]} \\
\text{destinationAirports, weights} &= \text{read_airports(}\text{destinationState}) \\
\text{dist} &= \left[ \frac{\text{float}(j[3])}{\text{distance_between_points_normal}(\text{person}[20], \text{person}[21], j[4], j[5])} \right]^{*2} \text{for } j \text{ in destinationAirports} \\
\text{weights} &= \text{classDumpModule5.cdf(dist)} \\
\text{split} &= \text{random.random()} \\
\text{idx} &= \text{bisect.bisect(weights, split)} \\
\text{airport} &= \text{destinationAirports[idx]} \\
\text{return } \text{airport} \\
\end{align*}
\]

\[
\begin{align*}
\text{def find_hotels(fips):} \\
\text{countyObject} &= \text{countyAdjacencyReader.read_data(}\text{fips}) \\
\text{allHotels} &= [] \\
\text{for } j \text{ in countyObject.neighbors:} \\
\text{hotels, stores} &= \text{get_hotses}(j) \\
\text{allHotels.append(j)} \text{for } j \text{ in hotels} \\
\text{return } \text{allHotels} \\
\end{align*}
\]

'Return List of Hotels and Stores in FIPS County'
```python
def get_hotels(fips):
    if len(fips) == 4:
        fips = '0' + str(fips)
    if fips == '15005':
        split = random.random();
        if split > 0.5: fips = '15003'
        else: fips = '15007'
    states = classDumpModule5.read_states()
    abbrev = classDumpModule5.match_code_abbrev(states, fips[0:2])
    filepath = dataRoot + 'Employment\CountyEmployeeFiles\' + abbrev + '\' + fips + '.\' + abbrev + '_EmpPatFile.csv'
    f = open(filepath, 'r+')
    reader = csv.reader(f, delimiter=',')
    stores = []; hotels = []
    for row in reader:
        code = row[8][0:2]
        if code == '72' or code == '62':
            stores.append(row)
        if row[8][0:4] == '7211':
            hotels.append(row)
    return hotels, stores

def distance_between_points_normal(lat1, lon1, lat2, lon2):
    degrees_to_radians = math.pi/180.0
    try:
        phi1 = (90.0 - float(lat1))*degrees_to_radians
    except ValueError:
        return 1000
    theta1 = float(lon1)*degrees_to_radians
    theta2 = float(lon2)*degrees_to_radians
    cos = (math.sin(phi1)*math.sin(phi2)*math.cos(theta1 - theta2) +
          math.cos(phi1)*math.cos(phi2))
    arc = math.acos(cos)
    distance = arc * 3963.167
    return distance

def execute(state, stateabbrev, county):
    global counties
    global stateCode
    global states
    stateCode = county[0:2]
    states = classDumpModule5.read_states()
    'Module 5 Input Path'
    fileLocation = rootFilePath + 'Module 5\' + state + '\' + state + '\' + county + '_' + inputFileNameSuffix
    'Module 5 MODAL Output Path'
    outputLocation = rootFilePath + 'Module 5 MODAL\' + outputFileNameSuffix
    startTime = datetime.now()
    'Open File'
    f = open(fileLocation, 'r')
    reader = csv.reader(f, delimiter=',')
    out = open(outputLocation, 'w+', encoding='utf8')
    personWriter = csv.writer(out, delimiter=',', lineterminator='
')
    counties = classDumpModule5.read_counties()
    stateAirports, weights = read_airports(stateabbrev)
    count = -1
    pCount = 0
    for p in reader:
        if count == -1: personWriter.writerow(p); count+=1; continue
        if p[6] == '-5':
            lat = (p[12])
            lon = (p[13])
            newP = send_to_international(lat, lon, p, stateAirports);
            personWriter.writerow(newP)
        elif p[9] == 'W':
```

116

lat = (p[12])
lon = (p[13])
workLat = float(p[20])
workLon = float(p[21])
distanceFirstTrip = distance_between_points_normal(lat, lon, workLat, workLon)

if distanceFirstTrip > 200.0:
    newP = take_a_plane(p, lat, lon, stateAirports)
    personWriter.writerow(newP)
else:
    personWriter.writerow(p)
pCount+=1

executive('NewYork', 'NY', '36061')

B.6 Module 6 Source Code

'''module6.py

Project: United States Trip File Generation – Module 6
Author: A.P. Hill Wyrough
version date: 3/15/2014
Python 3.3

Purpose: This program assigns all daily activity patterns their temporal attributes.
    It reads in old module 5 trip sequence files
and creates a new one with time attributes.

Dependencies: None

Notes: This is done entirely differently than mufti’s – it is adapted to more
    activity patterns and it uses exponential distributions
not triangular distributions as his did.

'PATH DEFINITIONS'"""
rootDrive = 'G'
rootFilePath = rootDrive + ':\Thesis\Output\'
inputFileNameSuffix = 'Module5NN1stRun.csv'
outputFileNameSuffix = 'Module6NN1stRun.csv'
dataDrive = 'G'
dataRoot = dataDrive + ':\Thesis\Data\'

import csv
import random
import math
from datetime import datetime
from os import listdir
from os.path import isfile, join

'ARRIVAL TIME PARAMETERS'
arrivalParameter = 5.0
departureParameter = 5.0
variance = 0.25
speedParameter = 30.0

'CONVERT MODULE 5 INPUT INTO MODULE 6 OBJECTS WITH TIME ADDED'
[['Type', 'Pred', 'Succ', 'Name', 'County', 'Lat', 'Lon', 'Industry', 'oTime', 'dTime']]
def initialize_new_activityPattern(activityPattern):
    pattern = []
c = 0
for j in range(0, 7):
activity = []
for k in range(0, 8):
    activity.append(activityPattern[c])
c+=1
activity.append('oTime')
activity.append('dTime')
pattern.append(activity)
return pattern

'WALK THROUGH TRIP SEQUENCE AND BUILD DAILY SCHEDULE'
'ACCEPTS OLD PERSONAL ACTIVITY PATTERN THEN RETURNS NEW ONE WITH TIME'
def scheduleDay(personalActivity, data, pattern):
    'HANDLE FIRST TRIP'
    home = personalActivity[0]
    home[8] = 0.0
    if home[2] == 'N':
        return personalActivity
    if home[2] == 'W':
        work = personalActivity[1]
        start, end, duration = get_employee_start_end_duration((work[7]), data)
        dTimeFirst = get_arrival_time(start)
        oTimeFirst = work_backwards(dTimeFirst, home[5], home[6], work[5], work[6])
        workDepartureTime = get_departure_time(end)
        home[9] = oTimeFirst
        work[8] = dTimeFirst
        personalActivity[1][9] = workDepartureTime
        personalActivity[2][8] = work_forwards(workDepartureTime, work[5], work[6],
                                              home[5], home[6])
    elif personalActivity[1][0] == 'S':
        school = personalActivity[1]
        dTimeFirst = get_arrival_time(8.5)
        oTimeFirst = work_backwards(dTimeFirst, home[5], home[6], school[5], school[6])
        schoolDepartureTime = get_departure_time(15.5)
        home[9] = oTimeFirst
        school[8] = dTimeFirst
        personalActivity[1][9] = schoolDepartureTime
        personalActivity[2][8] = work_forwards(schoolDepartureTime, personalActivity
                                              [1][5], personalActivity[1][6], home[5], home[6])
    elif personalActivity[1][0] == 'O':
        dTimeFirst = random.uniform(10, 12) * 3600
        oTimeFirst = get_arrival_time(8.5)
        dTimeFirst = work_forwards(oTimeFirst, home[5], home[6], personalActivity
                                    [1][5], personalActivity[1][6])
        duration = get_patronage_duration((personalActivity[1][7], data)
        firstPatronage = getDurationTime(duration)
        home[9] = oTimeFirst
        personalActivity[1][8] = dTimeFirst
        personalActivity[1][9] = dTimeFirst + firstPatronage
        personalActivity[2][8] = work_forwards(personalActivity[1][9],
                                              personActivity[1][5], personalActivity[1][6], personalActivity[2][5],
                                              personalActivity[2][6])
        if personalActivity[2][2] == 'N':
            return personalActivity
        if personalActivity[2][2] == 'O' and personalActivity[2][0] == 'H':
            personalActivity[2][9] = personalActivity[2][8] + random.uniform(0, 0.50) * 3600
            personalActivity[3][8] = work_forwards(personalActivity[2][9],
                                              personalActivity[2][5], personalActivity[2][6], personalActivity
                                              [3][5], personalActivity[3][6])
            personalActivity[3][9] = personalActivity[3][8] + getDurationTime(
                                      get_patronage_duration((personalActivity[3][7], data))
            personalActivity[4][8] = work_forwards(personalActivity[3][9],
                                              personalActivity[3][5], personalActivity[3][6], personalActivity
                                              [4][5], personalActivity[4][6])
    return personalActivity

'MANAGE LUNCH BREAK TRIPS (HWN)'
if pattern == '14':
    personalActivity[3][9] = workDepartureTime
forwards (departureTime, lat1, lon1, lat2, lon2):
    radians = math.pi / 180.0
    phi1 = (90.0 - float(lat1)) * degrees_to_radians
    phi2 = (90.0 - float(lat2)) * degrees_to_radians
    theta1 = float(lon1) * degrees_to_radians
    theta2 = float(lon2) * degrees_to_radians
    cos = (math.sin(phi1) * math.sin(phi2) + math.cos(theta1 - theta2))
    arc = math.acos(cos)
    return arc * 3963.167

actualArrivalTime = departureTime + tripTime
    if count == 6:
        if nextNode == 'H':
            duration = random.uniform(0, 0.30) * 3600.0
        else:
            duration = getDurationTime(get patronage_duration((now[7]), data))
        if nextNode == 'S':
            duration = getDurationTime(get patronage_duration(61, data))
        elif current == 'O':
            duration = random.gauss(4.00, 0.15 * 4.00) * 3600.0
        elif current == 'W':
            duration = getDurationTime(duration)
        if nextNode == 'N':
            now[9] = 'NA'
        break
    if count == 6:
        break
    next = personalActivity[count + 1]
    next[8] = forwards(now[9], now[5], now[6], next[5], next[6])
    count+=1
return personalActivity

'WALK THROUGH REMAINING TRIPS AND ASSIGN DURATION, ARRIVAL TIME'
for j in personalActivity[2:]:
def read_schedule_files():
    fileLocation = dataRoot + '\Trip Distributions and Times\ScheduleFile.csv'
    data = []
    f = open(fileLocation, 'r+')
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        data.append(row)
    return data

def get_patronage_duration(naics, data):
    if naics == 'NA' or naics == '99':
        naics = 81
    naics = int(naics)
    if naics in [31, 32, 33]: naics = 31
    if naics in [44, 45]: naics = 44
    if naics in [48, 49]: naics = 48
    for row in data:
        if int(row[1]) == naics:
            return float(row[5]) * 3600

def get_employee_start_end_duration(naics, data):
    if naics == 'NA' or naics == '99':
        naics = 81
    naics = int(naics)
    if naics in [31, 32, 33]: naics = 31
    if naics in [44, 45]: naics = 44
    if naics in [48, 49]: naics = 48
    for row in data:
        if int(row[1]) == naics:
            return float(row[2]), float(row[3]), float(row[4])

def get_arrival_time(bellTime):
    #COMPUTE ARRIVAL WINDOW TIME IN MINUTES
    windowTime = random.expovariate(1.0 / arrivalParameter)
    #COMPUTE ACTUAL ARRIVAL TIME IN SECONDS FROM MIDNIGHT
    secondsInWindow = 60.0 * windowTime
    arrivalTime = bellTime * 3600.0 - (10.0 * 60.0 - secondsInWindow)
    return arrivalTime

def get_departure_time(bellTime):
    #COMPUTE ARRIVAL WINDOW TIME IN MINUTES
    windowTime = random.expovariate(1.0 / arrivalParameter)
    #COMPUTE ACTUAL ARRIVAL TIME IN SECONDS FROM MIDNIGHT
    secondsInWindow = 60.0 * windowTime
    departureTime = bellTime * 3600.0 + secondsInWindow
    return departureTime

def writeHeaders(pW):
    pW.writerow(["Residence State"] + ["County Code"] + ["Tract Code"] + ["Block Code"] + ["HH ID"] + ["Person ID Number"] + ["Activity Pattern"]
    + ["Node 1 Type"] + ["Node 1 Predecessor"] + ["Node 1 Successor"] + ["Node 1 Name"] + ["Node 1 County"] + ["Node 1 Lat"] + ["Node 1 Lon"] + ["Node 1 Arrival Time"] + ["Node 1 Departure Time"]
    + ["Node 2 Type"] + ["Node 2 Predecessor"] + ["Node 2 Successor"] + ["Node 2 Name"] + ["Node 2 County"] + ["Node 2 Lat"] + ["Node 2 Lon"] + ["Node 2 Arrival Time"] + ["Node 2 Departure Time"]
    + ["Node 3 Type"] + ["Node 3 Predecessor"] + ["Node 3 Successor"] + ["Node 3 Name"] + ["Node 3 County"] + ["Node 3 Lat"] + ["Node 3 Lon"] + ["Node 3 Arrival Time"] + ["Node 3 Departure Time"]
    + ["Node 4 Type"] + ["Node 4 Predecessor"] + ["Node 4 Successor"] + ["Node 4 Name"] + ["Node 4 County"] + ["Node 4 Lat"] + ["Node 4 Lon"] + ["Node 4 Arrival Time"] + ["Node 4 Departure Time"]
    + ["Node 5 Type"] + ["Node 5 Predecessor"] + ["Node 5 Successor"] + ["Node 5 Name"] + ["Node 5 County"] + ["Node 5 Lat"] + ["Node 5 Lon"] + ["Node 5 Arrival Time"] + ["Node 5 Departure Time"]
    + ["Node 6 Type"] + ["Node 6 Predecessor"] + ["Node 6 Successor"] + ["Node 6 Name"] + ["Node 6 County"] + ["Node 6 Lat"] + ["Node 6 Lon"] + ["Node 6 Arrival Time"] + ["Node 6 Departure Time"]
def execute(filename, state):
    # READ INPUT
    inputFile = rootFilePath + ' Module 5 \\ + state + ' \\
    f = open(inputFile, 'r+')
    reader = csv.reader(f, delimiter=',')
    # CREATE OUTPUT
    splitter = filename.split('_')
    outputFile = rootFilePath + ' Module 6 \\
    o = open(outputFile, 'w+', encoding='utf8')
    personWriter = csv.writer(o, delimiter=',', lineterminator='
')
    # READ IN SCHEDULE FILE
    allTimes = read_schedule_files() 
    count = -1
    # SCHEDULE EVERYONES DAILY ACTIVITY PATTERN
    startTime = datetime.now()
    print(state + " started at: " + str(startTime))
    writeHeaders(personWriter)
    # ASSIGN EACH RESIDENT WITHIN THE COUNTY THEIR REVISED ACTIVITY PATTERN WITH NEW TIME ATTRIBUTES AND WRITE TO COUNTY FILE
    for r in reader:
        if count == -1: count += 1; continue
        activityPattern = r[6]
        personActivity = initialize_new_activityPattern(r[7:])
        personActivity = scheduleDay(personActivity, allTimes, activityPattern)
        count += 1
        if len(r[3]) == 4:
            r[3] = '0' + r[3]
            personWriter.writerow([r[0] + [r[1]] + [r[2]] + [r[3]] + [r[4]] + [r[5]] + [r[6]] + [personActivity[0][0]] + [personActivity[0][1]] + [personActivity[0][2]] + [personActivity[0][3]] + [personActivity[0][4]] + [personActivity[0][5]] + [personActivity[0][6]] + [personActivity[0][7]] + [personActivity[1][0]] + [personActivity[1][1]] + [personActivity[1][2]] + [personActivity[1][3]] + [personActivity[1][4]] + [personActivity[1][5]] + [personActivity[1][6]] + [personActivity[1][7]] + [personActivity[2][0]] + [personActivity[2][1]] + [personActivity[2][2]] + [personActivity[2][3]] + [personActivity[2][4]] + [personActivity[2][5]] + [personActivity[2][6]] + [personActivity[2][7]] + [personActivity[2][8]] + [personActivity[2][9]] + [personActivity[3][0]] + [personActivity[3][1]] + [personActivity[3][2]] + [personActivity[3][3]] + [personActivity[3][4]] + [personActivity[3][5]] + [personActivity[3][6]] + [personActivity[3][7]] + [personActivity[3][8]] + [personActivity[3][9]] + [personActivity[4][0]] + [personActivity[4][1]] + [personActivity[4][2]] + [personActivity[4][3]] + [personActivity[4][4]] + [personActivity[4][5]] + [personActivity[4][6]] + [personActivity[4][7]] + [personActivity[4][8]] + [personActivity[4][9]] + [personActivity[5][0]] + [personActivity[5][1]] + [personActivity[5][2]] + [personActivity[5][3]] + [personActivity[5][4]] + [personActivity[5][5]] + [personActivity[5][6]] + [personActivity[5][7]] + [personActivity[5][8]] + [personActivity[5][9]] + [personActivity[6][0]] + [personActivity[6][1]] + [personActivity[6][2]] + [personActivity[6][3]] + [personActivity[6][4]] + [personActivity[6][5]] + [personActivity[6][6]] + [personActivity[6][7]] + [personActivity[6][8]] + [personActivity[6][9]] + [personActivity[7][0]] + [personActivity[7][1]] + [personActivity[7][2]] + [personActivity[7][3]] + [personActivity[7][4]] + [personActivity[7][5]] + [personActivity[7][6]] + [personActivity[7][7]] + [personActivity[7][8]] + [personActivity[7][9]] + [personActivity[8][0]] + [personActivity[8][1]] + [personActivity[8][2]] + [personActivity[8][3]] + [personActivity[8][4]] + [personActivity[8][5]] + [personActivity[8][6]] + [personActivity[8][7]] + [personActivity[8][8]] + [personActivity[8][9]] + [personActivity[9][0]] + [personActivity[9][1]] + [personActivity[9][2]] + [personActivity[9][3]] + [personActivity[9][4]] + [personActivity[9][5]] + [personActivity[9][6]] + [personActivity[9][7]] + [personActivity[9][8]] + [personActivity[9][9]]

121
perActivity[4][8] + [perActivity[4][9]] + 
[perActivity[5][0]] + [perActivity[5][1]] + [perActivity[5][2]] + [perActivity[5][3]] + [perActivity[5][4]] + [perActivity[5][5]] + [perActivity[5][6]] + [perActivity[5][8]] + [perActivity[5][9]] + 
[perActivity[6][0]] + [perActivity[6][1]] + [perActivity[6][2]] + [perActivity[6][3]] + [perActivity[6][4]] + [perActivity[6][5]] + [perActivity[6][6]] + [perActivity[6][8]] + [perActivity[6][9]]

if count % 100000 == 0:
    print(str(count) + ' Residents Completed and taken this much time:
                     ' + str(datetime.now() - startTime))
    f.close()
o.close()
print(str(count) + ' of all Residents in ' + filename + ' have been processed')
print(filename + ' took this much time: ' + str(datetime.now() - startTime))

'ASSIGN ALL TRIP TOURS THEIR TEMPORAL ATTRIBUTES'
'RUN BY STATE, WHICH READS ALL COUNTY FILES IN STATE FOLDER'
def state_run(state):
    mypath = rootFilePath + ' Module 5' + state + '/'
    onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
    for j in onlyfiles:
        executable(j, state)

import sys
exec('state_run(sys.argv[1])')
Bibliography


InfoGroup (2014). ReferenceUSA Business Database. Online Database.


Murray, P. (2012). Google’s Self-driving Car Passes 300,000 Miles. [Online; posted 15-Aug-2012].


