5-2010

Understanding and Minimizing Flight Delay

Patrick Robert Steele
College of William and Mary

Understanding and Minimizing Flight Delay

A thesis submitted in partial fulfillment of the requirement
for the degree of Bachelor of Science in Mathematics from
The College of William and Mary

by

Patrick Robert Steele

Accepted for _____

_____

Tanujit Dey, co-director

_____

David Phillips, co-director

_____

Rex Kincaid

_____

Larry Leemis

_____

Carlisle Moody

Williamsburg, VA
April 28, 2010

Contents

Acknowledgments

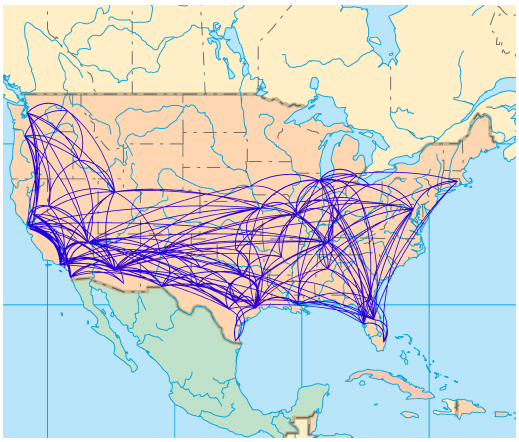
This project was partially finded by the Charles Center at William and Mary through a Chappell Fellowship for the summer of 2009. I would also like to thank the CSUMS group and the Department of Mathematics for year round support.

I would like to thank my advisors, Professors Tanujit Dey and David Phillips, for their long hours of advice and support while working on this project. I would also like to Professors Larry Leemis and Rex Kincaid of the Mathematics department and Carlisle Moody of the Economics department for serving on my honors committee and offering insightful advice and criticism on my work. Finally, I would like to thank Tom Crockett for providing outstanding support for the SciClone computing network.

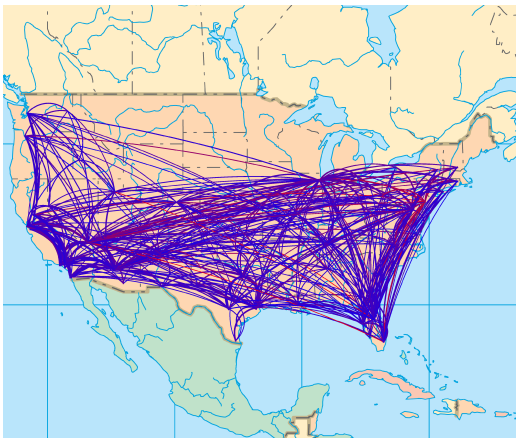
1987

1997

2002



2008



The expansion of Southwest Airlines, from 1987 to 2008.

Abstract

We model the problem of finding minimum-delay routes through the U.S. flight network as a shortest paths problem with stochastic costs. Our approach utilizes statistical modeling, Monte Carlo simulation, and network optimization techniques. Our models and simulations are based on data obtained from the U.S. Department of Transportation. Our sampling techniques allow us to model the time dependence of arc costs.

In chapter 1, we introduce the problem, previous approaches, and our methods. In chapter 2, we give an overview of the data set that we work with. In chapter 3, we introduce the statistical techniques that we use to model flight delay. In chapter 4, we introduce the delay minimization problem as a stochastic optimization problem, as well as techniques to solve it. Finally, in chapter 5 we utilize our methods to solve a variety of problems, as well as demonstrate the graphical capabilities of our software. Appendix A contains graphics summarizing the data set we used, while appendix B contains the parameter values discussed in chapter 3 that were used in our simulations.

Chapter 1

Introduction

*"The total cost of domestic air traffic delays to the U.S. economy was as much as **$41 billion** for 2007."*

– U.S. Congress Joint Economic Committee

1.1 Background

The U.S. flight network is a complex system; one major carrier, American Airlines, flew over 585,000 flights in the year 2008 alone.[1] It comes as no surprise that understanding the causes and behavior of flight delay in this system is a difficult task; however, this is a topic of interest to both travelers and industry. The ability to predict flight delay would allow travelers to make better informed decisions when making travel plans, while the ability to identify the causes of delay at problem airports would allow airlines to improve their services. We constructed a tool that utilizes techniques from statistics and stochastic optimization to find flight paths with the minimum expected delay.

Our first goal was to develop a greater understanding of the causes and behavior of flight delay in the United States airline network; we then used this information to predict flight

[1] Calculated from the data provided by the U.S. Department of Transportation's Bureau of Transportation Statistics.

delay. We wished to determine which variables are likely to increase or mitigate flight delay. We also wished to be able to answer questions such as which airlines or airports experience more or less delay. To accomplish this, we utilized statistical techniques such as multiple linear regression and least-squares estimation to determine a stable estimator of the data.

Our second goal is to develop techniques to find delay minimizing flight paths between given origin and destination cities, subject to any number of constraints; for example, we wished for our techniques to be able to find the flight path with the minimum expected delay given that the flight was flown by Delta airlines on a Wednesday in September.

To accomplish this, we formulated the delay-minimization problem as a linear integer program with random lengths. We devised and implemented an algorithm to solve the linear stochastic program, and generalized the problem to minimizing a larger class of weights, such as total flight time.

Since flight delay is a stochastic quantity, we cannot determine a closed-form solution to the minimization problem; instead, we used Monte Carlo simulation to approximate the stochastic network, solving the deterministic minimization problem each time. The Monte Carlo simulation utilizes statistical techniques to predict the behavior of networks. The deterministic minimization problem was solved using results from graph theory. Algorithm 5 outlines these methods.

1.1.1 Literature Review

Previous work has generally focused on decisions at the level of airline management. Bratu and Barnhart [2] consider the problems faced by airlines encountering unexpected delays, such as inclement weather, mechanical failure, and personnel problems. They utilize integer programming techniques to solve the scheduling problem of minimizing delay and maximizing airline profit when faced with these issues. Delahaye and Odoni [4] consider the problem of minimizing airspace congestion through stochastic optimization techniques;

given a set of flight plans to be met, they consider the problem of minimizing the amount of traffic through air sectors and airports throughout the day. Nilim, El Ghaoui, and Duong [6] consider the problem of routing multiple aircraft in the presence of inclement weather patterns through the use of Markov chains. Yan and Chen [8] simulate the effectiveness of static and real-time gate assignment in airports to improve the efficiency of airports. Lan, Clarke and Barnhart [5]have used mixed-integer programming to route aircraft to avoid propagated delay, as well as to minimize the number of passengers who miss connections because of insufficient transfer time. Argüello, Bard, and Yu [1] explore the use of a greedy randomized search for adaptively reconstructing flight routes when delay or cancellation interrupts normal activity. We discuss the shortest paths problem in detail in section 1.3

Our methods focus on providing a tool that is useful to both individual travelers and airlines. Our methods do not involve re-routing flight paths or changing the behavior of the network in any way — both activities beyond the control of a single traveler — and so are applicable to individuals and industry alike.

1.1.2 Notation

We assume a basic familiarity with algorithms on the level of [3]. We make use of the following notational conveniences, for which we draw strongly from [3]; we define many of these terms in more detail in sections 1.2 and 1.3.

Graphs

- $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ a graph with vertexes $\mathcal{V}$, edges $\mathcal{E}$, and weights $\mathcal{W}$.
- $(a,b) \in \mathcal{E}$ is the edge from a to b in $\mathcal{E}$.
- k-path. A path in a graph containing k or fewer arcs.
- $a \to b$, or an (a,b)-path, is a path from a to b.

- $a \xrightarrow{k} b$ is a k-path from a to b.
- $(a_1, a_2, \ldots, a_{k+1})$ is an ordered sequence of nodes to traverse to form the k-path from $a_1 \to a_{k+1}$.

Algorithms

- *italic* words are variables,
- **bold** words are language, constructs
- `monospace` words are algorithm names.

1.2 Graph Theory

For completeness, we now offer a brief introduction to graph theory; we base our approach on the work of Cormen, Leiserson, Rivest, and Stein [3].

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a set of vertices, $\mathcal{V}$, and edges, $\mathcal{E}$; we define $|\mathcal{V}| = n$ and $|\mathcal{E}| = m$. Each edge connects two vertices in $\mathcal{V}$; for nodes $a, b \in \mathcal{V}$, the notation $(a, b) \in \mathcal{E}$ indicates that there is an edge from a to b in $\mathcal{G}$. We also refer to vertices and edges as nodes and arcs, respectively. It is important to note that a graph does not need to contain all possible edges; in fact, the set $\mathcal{E}$ is allowed to be empty.

A graph may be *directed* or *undirected*. We say that $\mathcal{G}$ is a directed graph if for all edges $(a, b) \in \mathcal{E}$, (a, b) is an ordered pair.

A graph may be *weighted* or *unweighted*. In a weighted graph, each edge $(a, b) \in \mathcal{E}$ has an associated weight $w_{a,b}$, where $w_{a,b} \in \mathbb{R}$. An unweighted graph has $w_{a,b} = 1$ for all $(a, b) \in \mathcal{E}$. We say that a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ has vertices $\mathcal{V}$, edges $\mathcal{E}$, and associated edge weights $\mathcal{W}$. Weights in a graph may be positive or negative.

A *walk* in a graph is a sequence of nodes $u = a_0, a_1, \ldots, a_k \in \mathcal{V}$ such that the edges $(a_0, a_1), (a_1, a_2), \ldots, (a_{k-1}, a_k)$ are all in $\mathcal{E}$, where k is a positive integer. For given nodes

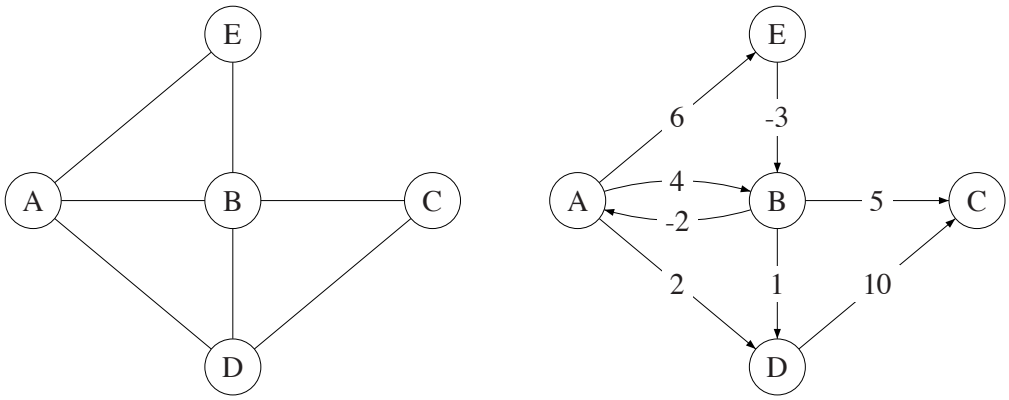


Figure 1.1: Examples of graphs. On the left is an undirected, unweighted graph, and on the right is a weighted, directed graph.

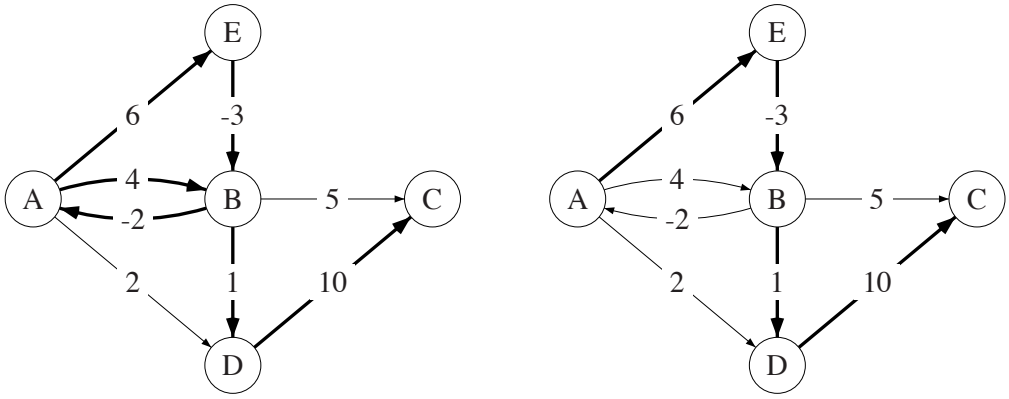


Figure 1.2: Examples of walks and paths. On the left, the bold edges indicate a walk from node A to node C ($A-B-A-E-B-D-C$), while on the right the bold edges indicate a path from node A to node C ($A-E-B-D-C$).

a,b, we define an (a,b)-path as a walk $a,\ldots,b$. A closed walk has $a_0 = a_k$; a *cycle* is a closed walk with only one node repeated. Define

$$w(u) = \sum_{(a,b)\in u} w_{a,b} \tag{1.1}$$

as the weight of a walk u; then a negative cycle is a cycle u with $w(u) < 0$.

Graphs are often used to represent networks of various sorts. An example of an undirected graph would be a map of the countries of the world. We could assign each country a node, and then draw an edge between two country's nodes if those countries share a bor-

der. This model naturally uses undirected arcs, because if a country shares a border with another, the other necessarily shares one with it. Note that we could still assign weights to the edges. For example, we could use a $\{-1,0,1\}$ indicator variable to represent two countries diplomatic relations; if two countries are hostile, we could assign a value of -1, while if they are neutral or allied, we could assign a weight of 0 or 1, respectively.

Another example is the U.S. highway system. By defining each intersection as a node and each road segment as an arc, we can represent the system as a graph. We use a directed graph for this model since not all roads are two-way. So far, this model represents the relationships between each intersection, and could be used for rudimentary navigation. However, what is not modeled is the distance between intersections. To accomplish this, we add weights to the graph, using road lengths as arc weights. Now, the graph represents both the relationships between intersections and the distance between them. Alternatively, we could use the expected time of travel as arc weights, computed by dividing the road length by the posted speed limit.

In the previous example, a natural question that arises is the following: what is the fastest way to travel between two points in the graph? In the previous example, 'fastest' could either refer to the shortest distance or the shortest amount of time; in the case of the former, we would use road lengths as weights, while in the case of the latter, we would use road lengths divided by posted speed limits. Questions of this form are known as *shortest path problems*.

1.3 The Shortest Path Problem

Consider two vertices a,b in a weighted graph $\mathcal{G} = (\mathcal{V},\mathcal{E},\mathcal{W})$. The *single pair shortest path problem* finds the path $p = a,a_1,\ldots,a_k,b$ such that

$$w(p) \leq w(q),$$

for all (a,b) paths in $\mathcal{G}$. Thus, the shortest path problem attempts to find the weight-minimizing path between two nodes in the graph. If no (a,b) path exists in $\mathcal{G}$, the shortest path weight is defined as positive infinity.

Several interesting questions arise immediately when considering the shortest path problem. Foremost, we can ask whether such a path even exists. Clearly, for a shortest path to exist, some path must exist, and so the nodes of interest must be connected. Second, we note that the graph must not contain any negative cycles that are reachable from our origin node. If a graph were to contain such a cycle, then for any path weight w between the origin and destination, one can find a path of weight $w' < w$ simply by navigating the negative cycle a large enough number of times. Note that a sufficient condition for a graph to contain no negative cycles is for all weights in the graph to be nonnegative.

We generalize the single pair shortest path problem to the *single source shortest path problem*. In this problem, we consider finding the set of shortest paths between the origin and every other node in the graph. In practice, algorithms that efficiently solve the single pair problem also solve the single source problem; briefly, if $a_0, a_1, \ldots, a_k$ is the shortest path to a_k, then $a_0, a_1, \ldots, a_i$ is the shortest path to a_i, where i,k are positive integers with $i < k$, and so in the worst case the single pair problem actually solves the single source problem.

1.3.1 The Single Source Shortest Path Problem

The objective of the single-source shortest path problem is to compute the *shortest paths tree* from the specified origin node n_o over a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$. A *tree* is a graph in which each node has no more than one incident and one outgoing arc; the shortest paths tree for $\mathcal{G}$ is a tree formed from the vertices $\mathcal{V}$, where each path from the origin node n_o to any other node is the shortest path between the two. Figure 1.3 demonstrates a solution to the single-source shortest path problem.

For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, we define the *predecessor* function $\pi : \mathcal{V} \to \mathcal{V}$, as

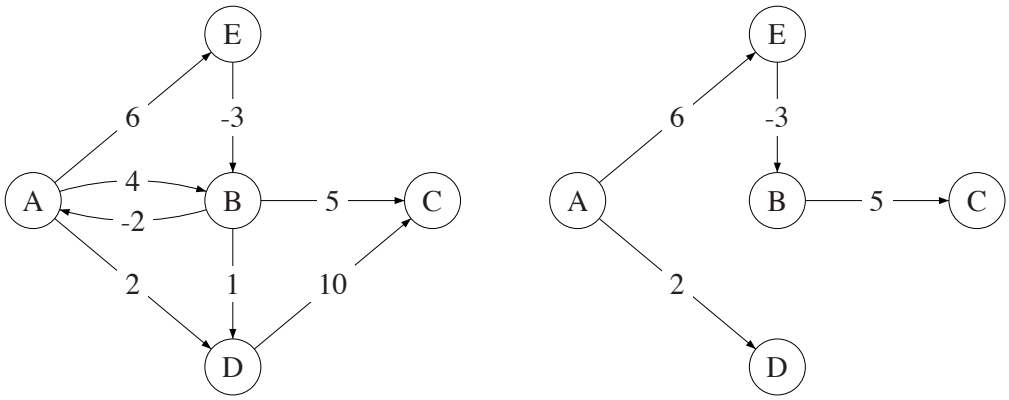


Figure 1.3: A solution to the single-source shortest paths problem. On the left is a weighted graph, and on the right is the shortest paths tree with node A as the origin. Note how no node can reached by more than one arc.

mapping a vertex to its parent in the shortest paths tree; we define $\pi(n_o) = n_o$. We also define the *depth* function $d : \mathcal{V} \to \mathbb{R}$ as the weight of the shortest path between the origin node and another node in the tree. Note that for the π and d functions we will use the notation $\pi(a) \leftarrow b$ and $d(a) \leftarrow x$ to indicate that $\pi(a) = b$ and $d(a) = x$, respectively.

Note that if the values of π and d were known for all vertices in $\mathcal{V}$, then we could construct the shortest path between the origin and any node i in the graph by utilizing algorithm 1 below:

Algorithm 1 An algorithm used to recover the shortest path between the origin and any other node in a graph and its weight, using the variables π and d computed beforehand.

Define: `backsolve`
Require: π, d, origin i_o, destination i_d
  $i \leftarrow i_d$
  $path \leftarrow \text{list}(i_d)$
  repeat
    $i \leftarrow \pi(i)$
    append i to $path$
附加 **until** $i = i_o$
  reverse $path$
  return $path, d(i_d)$

We first consider the single-source shortest path problem on an unweighted graph; here, we consider each arc to be of equal length, and so the distance between two nodes is simply

defined as the minimum number of arcs needed to traverse between them. This problem is solved via the *breadth-first search* technique, a graph discovery technique.

1.3.2 Breadth-First Search

The breadth-first search technique is an algorithm used to discover all nodes in a graph that can be reached from a given origin node, as well as determine the minimum number of arcs separating the origin node and each other node. The key characteristic of the breadth-first search algorithm is that it attempts to search all nearby nodes before searching further away; this results in the algorithm discovering nodes in order of distance, with the closest nodes being discovered first. Note that this search ignores arc weights. The breadth-first search algorithm is shown in figure 1.4.

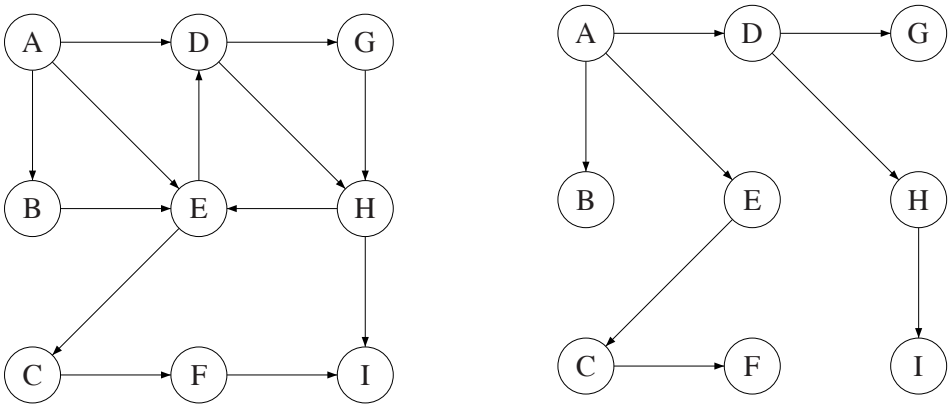


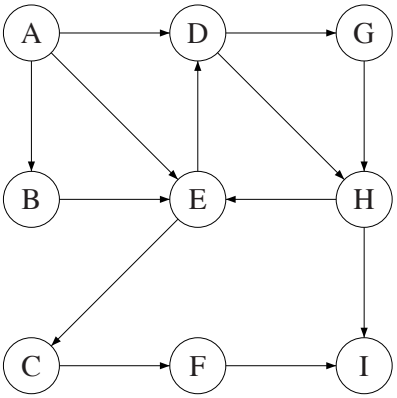
Figure 1.4: An example of a breadth-first search. On the left is our original graph, and on the right is the shortest paths tree formed by a breadth-first search from node *A*. Note how no node is reached from more than one arc, and that we do not consider the weights of the arcs (not shown). The shortest-paths tree allows us to see that, for example, node *I* is, at a minimum, 3 arcs from *A*. Note that the minimum number of arcs separating the origin and any node *n* and the path between them can be found via `backsolve`.

The breadth-first search algorithm is a useful tool, and will be utilized later. However, it does not account for arc weights, and so does not solve the general single-source shortest path problem. To accomplish this, we utilize Dijkstra's algorithm, a well-known, efficient algorithm for solving this problem.

Algorithm 2 The breadth-first search algorithm is used to determine the minimum number of arcs needed to traverse from a specified origin node to each destination node. This algorithm makes use of a graph coloring argument; nodes labeled 'white' have not been visited, nodes labeled 'gray' have been visited, but not explored, and nodes labeled 'black' have been visited and explored.

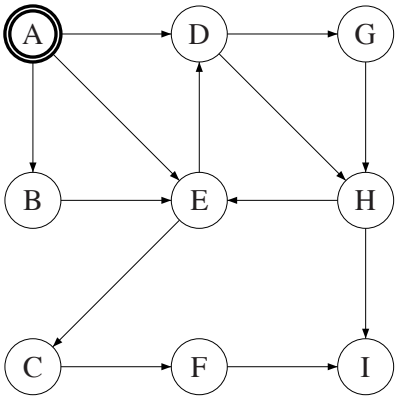
Define: `breadth-first-search`
Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, origin i_o
  for v **in** $\mathcal{V}$ **do**
    $\pi(v) \leftarrow v$
    $d(v) \leftarrow \infty$
    $c(v) \leftarrow$ white
  end for
  $d(i_o) \leftarrow 0$
  append i_o to an empty FIFO queue Q
  while Q not empty **do**
    $i \leftarrow$ next in Q
    remove n from Q
    $c(i) \leftarrow$ black
    for $v \in V$ adjacent to n **do**
      if $c(v)$ is white **then**
        add v to end of Q
        $\pi(v) \leftarrow i$
        $d(v) \leftarrow d(i)+1$
        $c(v) \leftarrow$ gray
      end if
    end for
  end while
  return π, d

Queue



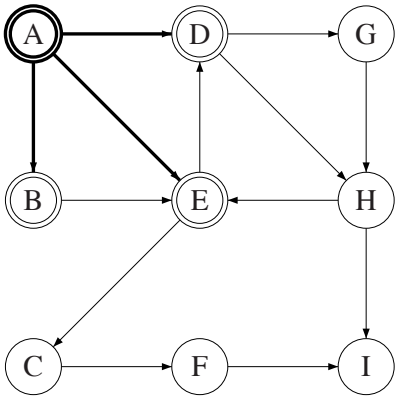
The original graph. We will perform a breadth-first search from A.

Queue A



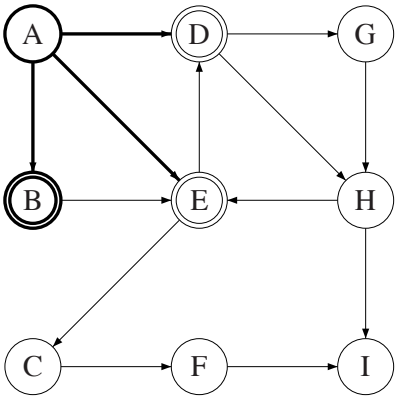
We add A to the queue.

Queue A B D E



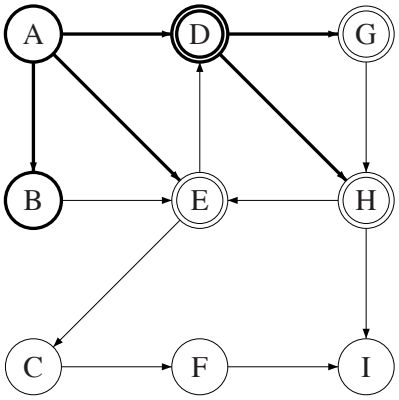
A has three undiscovered neighbors, B, D, and E; we add them to the queue.

Queue B D E



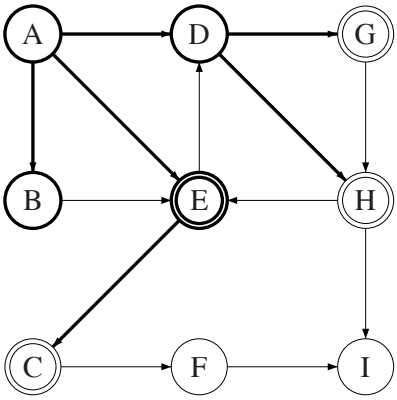
We remove A from the queue, and consider B. B has no undiscovered neighbors, so no nodes are added to the queue.

Queue D E G H



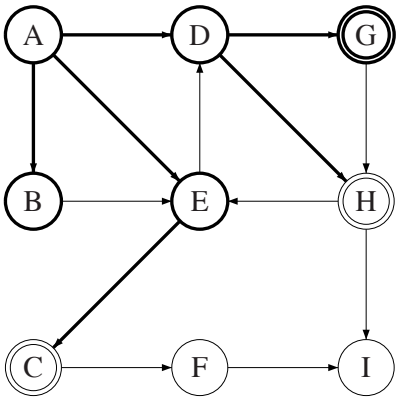
We remove B, and consider D. D has two undiscovered neighbors, G and H. We add them to the queue.

Queue E G H C



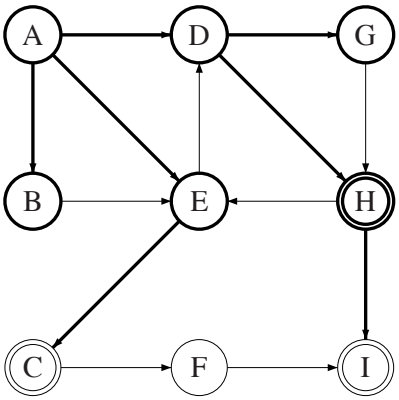
We remove D from the queue and consider E. E has one undiscovered neighbor, C; we add it to the queue.

Queue G H C



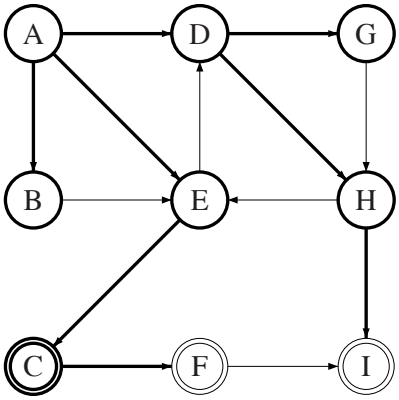
The next node, G, has no undiscovered neighbors. No nodes are added to the queue.

Queue H C I



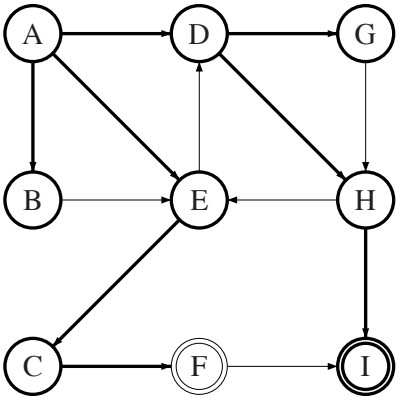
H has one undiscovered neighbor, I, which is added to the queue.

Queue C I F



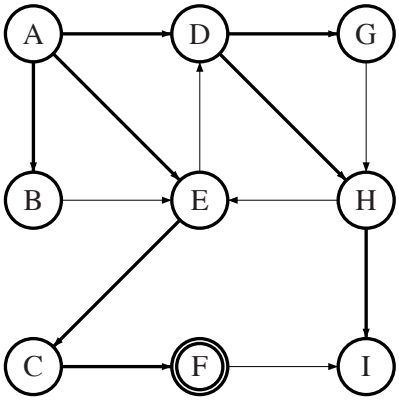
C has one undiscovered neighbor, *F*, which is added to the queue.

Queue I F



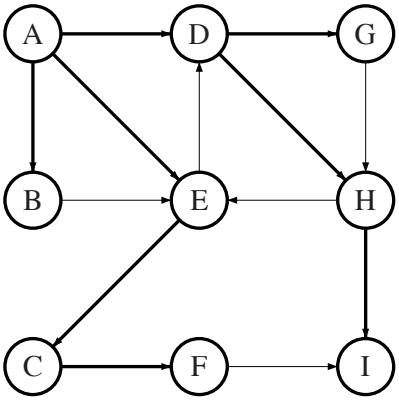
I has no undiscovered neighbors. No nodes are added to the queue.

Queue F



F has no undiscovered neighbors. No nodes are added to the queue.

Queue



The queue is now empty, so the search is complete.

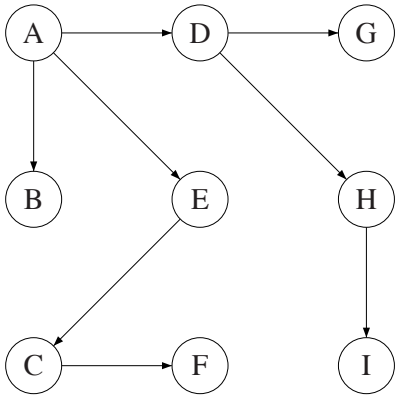


Figure 1.5: The final breadth-first search tree, after removing all arcs that were not traversed in finding undiscovered nodes.

1.3.3 Dijkstra's Algorithm

In order to utilize Dijkstra's algorithm, we must have a graph with no negative arc weights; we assume this to be true. In practice, we simply define $w_{a,b} = 0$ should our methods predict a negative delay.

Before we introduce Dijkstra's algorithm, we first consider the algorithm `relax` below. Given functions π, d, and w, all with some initial values, tail node t, and head node h, `relax` attempts to provide a better estimate on the distance to the head node h; that is, if the cost of the path from the origin to t and then to h, $d(t) + w_{t,h}$, is less than the current best estimate of the distance to h, $d(h)$, then the predecessor of h and the distance $d(h)$ are updated to reflect this. We now consider Dijkstra's algorithm, which makes use of `relax`.

Algorithm 3 This algorithm attempts to reduce the best known estimate for the value of d for a given vertex. If the estimate is changed, then the corresponding value of π is updated to reflect this.

Define: `relax`
Require: π, d, w, tail t, head h
  if $d(h) > d(t) + w(t,h)$ **then**
    $\pi(h) \leftarrow t$
    $d(h) \leftarrow d(t) + w(t,h)$
  end if

Dijkstra's algorithm computes the values of the π and d functions by maintaining best-estimates of each. At each iteration of the algorithm, it selects the node with the lowest estimated distance from the origin that has not yet been selected, and calls `relax` on all outgoing arcs from it. Dijkstra's algorithm is presented in full in algorithm 4.

Algorithm 4 Dijkstra's algorithm is used to solve the single-source shortest path problem for weighted, nonnegative graphs from some origin node n_o.

Define: `dijkstra`
Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, cW)$, origin n_o
  for v **in** $\mathcal{V}$ **do**
    $\pi(v) \leftarrow v$
    $d(v) \leftarrow \infty$
    $c(v) \leftarrow$ white
  end for
  $d(n_o) \leftarrow 0$
  set $Q \leftarrow \mathcal{V}$
  while Q not empty **do**
    $n \leftarrow \min Q$
    remove n from Q
    for $v \in V$ adjacent to n **do**
      `relax`$(\pi, d, w,$ tail $n,$ head $v)$
    end for
  end while
  return π, d

1.4 The Model

To better understand the U.S. flight network, we first created a model with which to view the network. At its most basic level, the flight network is extraordinarily large – 1,186,911 flights for Southwest Airlines in 2008 alone – as well as high-dimensional; each flight in the network has over twenty parameters associated with it. To work with the flight network in this form would be difficult at best; to improve our chances of finding meaningful results, we cast the problem into a more illuminating form. We note that the structure of the flight network motivates a representation using the graph theory we have discussed above; to accomplish this, we assigned each airport in the network a node, and each flight an arc.

The tail of each flight is the origin airport, and the head is the destination airport; we also assign a vector weight to each arc consisting of all the data we have on that flight.

However, this model is not quite complete. The U.S. flight network is not time-independent; the availability of flights changes throughout the day, as well as from day to day, making it a *dynamic network*. Thus, there may be more than one arc between two nodes, where each additional arc represents a flight from a different time.

We wished to consider the problem of finding the delay-minimizing path between two airports in the network. Allow the flight network to be represented as the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, where $\mathcal{W}_{ij}$ represents the total delay between airport i and j. Then the problem can be expressed as the following linear integer program

$$\min \left\{ \sum_{(i,j)\in\mathcal{E}} \mathcal{W}_{ij}\gamma_{ij} : \gamma_{ij} \in \Omega \right\} \tag{1.2}$$

where

$$\Omega = \left\{ \gamma : \underbrace{\sum_{(i,j)\in\varepsilon} \gamma_{ij}}_{\text{Flights out of } i} - \underbrace{\sum_{(j,i)\in\mathcal{E}} \gamma_{ij}}_{\text{Flights into } i} = b_i, \forall i \in \mathcal{V} \right\}, \tag{1.3}$$

and $\gamma_{ij} \in \{0,1\}$ represent yes-or-no *decision variables* representing whether arc (i,j) is included in the chosen path, and $b_i = -1$ if i is the origin node, $b_i = 1$ if i is the destination node, and $b_i = 0$ otherwise. Thus, the first equation simply finds the sum of the weights of all included paths, and attempts to minimize this quantity. It is restricted by the constraint $\gamma_{ij} \in \Omega$. The set Ω is the set of all possible paths that satisfy *flow balance constraints*; that is, it is the set of all paths such that exactly one arc leaves the origin, exactly one arc enters the destination, and every other node has an equal number of incoming and outgoing arcs. Additionally, we note that we know very little about the set $\mathcal{W}$; in fact, we assume that

$$w_{i,j} \sim D_{i,j}, \quad \forall (i,j) \in \mathcal{E},$$

where $D_{i,j}$ is some unknown distribution.

Clearly, if we were able to assign values to the elements of $\mathcal{W}$, then we could solve equation (1.2) utilizing the techniques outlined in section 1.3; however, we were unable to do so because the elements of $\mathcal{W}$ are distributed randomly according to unknown probability distribution functions.

As such, we were unable to find a solution to the problem as it was; we required additional information to be able to assign values to the elements of the set $\mathcal{W}$. To accomplish this, we utilized Monte Carlo simulation and multiple linear regression on our data sets in order to predict the values of the elements of $\mathcal{W}$. Briefly, our methods were as follows:

Algorithm 5 A basic outline of our methods. We discuss the parameters from line one in chapter 2, while in chapter 4 we discuss sampling and simulation techniques. Finally, we discuss the graphical representation of the results in chapter 5.

Require: parameters (carrier, desired dates, ...)
  repeat
    sample relevant data
    simulate all relevant arc costs
    solve the deterministic shortest path problem on the simulated network
  until the standard error of the prediction is small
  report results graphically

We will discuss these techniques in detail in chapter 3, but for now we will concern ourselves with understanding the data set with which we are working.

Chapter 2

The Data

2.1 Source

The data set used came from the U.S. Department of Transportation's Bureau of Transportation Statistics, and includes data on all domestic flights from 1987 to 2008. Each record in the dataset represents a single flight, and contains values for the following fields: the day of week, day of month, month, and year of the flight, the arrival and departure times, the carrier, the taxi in and taxi out times, the origin and destination airport, the distance traveled, the time in the air, cancellation status, and the arrival, departure, security, National Air System (NAS) [**?**], weather, and carrier delay.

With such a large dataset, it is difficult to see any trends simply by browsing the data; the file containing the records for American Airlines flights from 2005 to 2008 alone contains 2,493,190 lines of data. To overcome this, we turned to statistical analysis.

2.2 Overview of the Data

We began our study of the U.S. flight network using simple statistics. First, we considered the mean delay across all flights by each carrier, and found that that American Airlines flights have an average of 37 minutes of delay, while Continental flights average 35 min-

utes, Delta flights average 26 minutes, American Eagle flights average 34 minutes, SkyWest flights average 25 minutes, United flights average 37 minutes, and Southwest flights average 24 minutes. Next, we considered the variance of the delay, and found that each carrier has delay variances in the thousands of minutes. This is to be expected; since we were dealing with a large sample size and an essentially stochastic process, we expected there to be a large number of heavy outliers. Additionally, it is not uncommon for flights delayed due to weather to be delayed for days, yielding tremendously large delay times. Table A.1 summarizes the mean, variance, and standard deviations of the total flight delay of each carrier. However, it is difficult to visualize the distribution of delays from the mean and variance alone; to overcome this, we created a histogram of the frequency of delay for each airline. Figure 2.1 shows the distribution of total flight delay times for American Airlines.

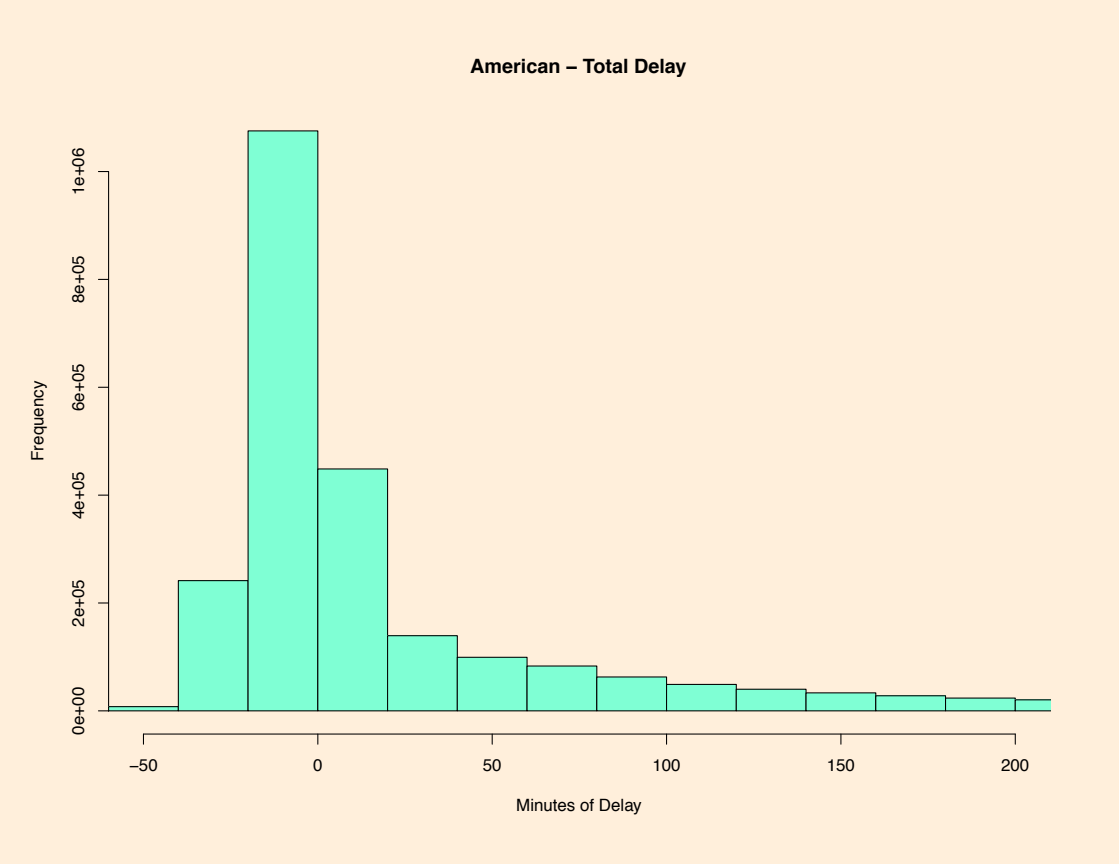


Figure 2.1: A histogram plotting the frequency of total delay time. The data is for American Airlines from 2005–2008.

Histograms of delay times for each carrier can be found in appendix A. However, these histograms do not provide an easy way to compare the delay between all seven carriers; to accomplish this, we created box-and-whisker plots displaying the five number statistics of delay time for each carrier; figure 2.2 shows this plot. As can be seen, box-and-whisker plots nicely summarize the differences between carriers, but leave out the detail of the histograms from before.

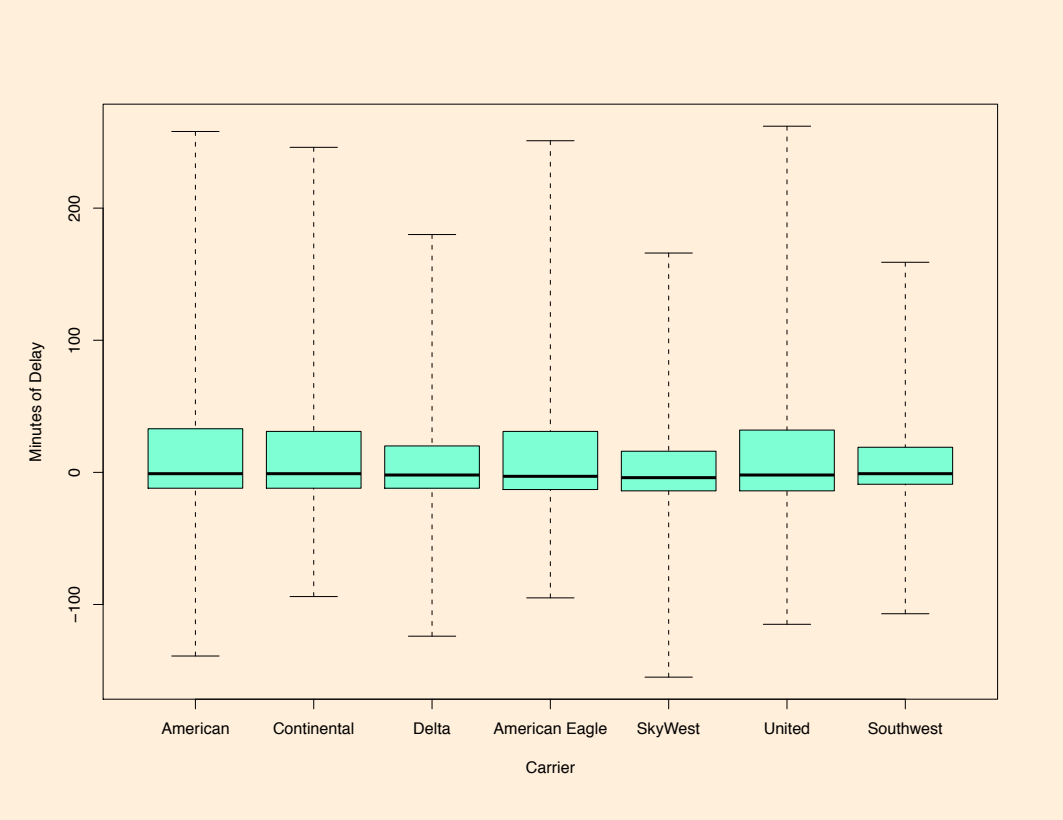


Figure 2.2: A box-and-whisker plot of the total delay for each major carrier. The data is from 2005–2008. Negative times indicate early arrivals.

Next, we considered the various components of delay that contribute to the total delay. We find the sample mean, variance, and standard deviation of each component of delay. These values can be found in appendix A.1. Since we have 7 components of delay, these statistics would produce 21 numeric values per carrier, which is difficult to visualize. First, we created histograms of each component of delay to visualize the distributions, as before.

Figure 2.3 shows the distribution of arrival delay times for American Airlines.

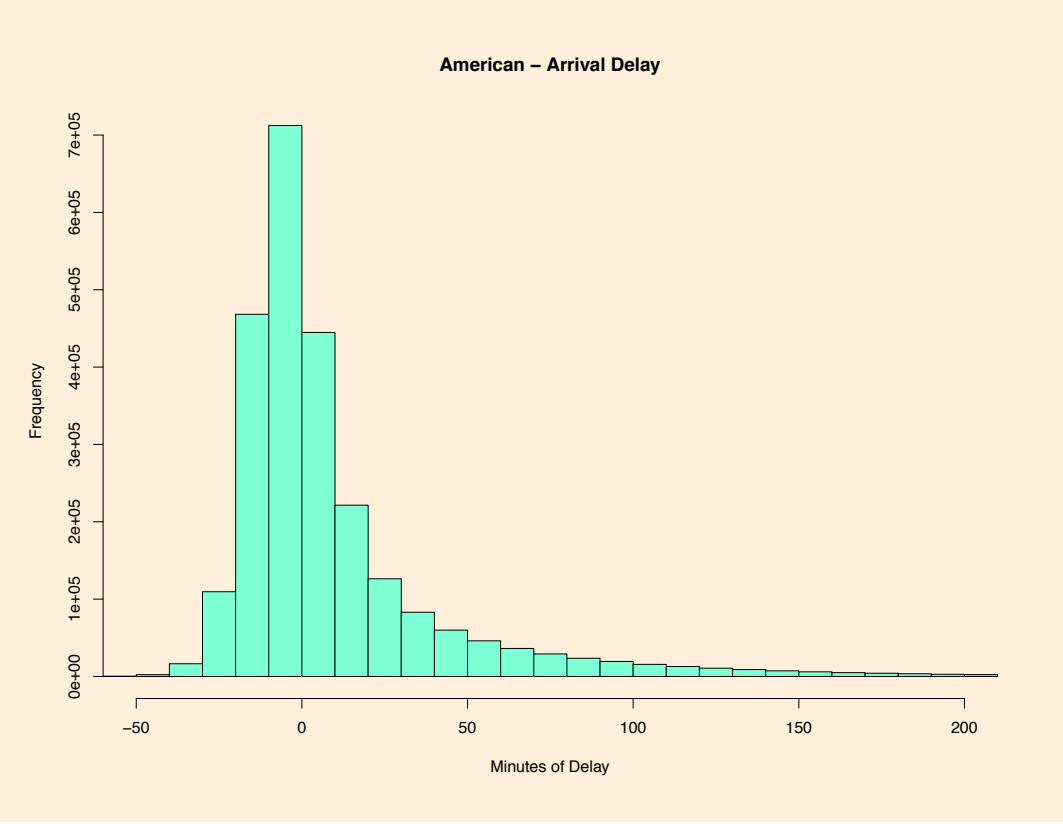


Figure 2.3: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

Histograms of each component of delay for each carrier can be found in appendix A. However, like before, these histograms do not provide an easy way to compare the distributions of the seven kinds of delay to each other. Again, we created box-and-whisker plots displaying the five number statistics of each component of delay for each carrier. Figure 2.4 shows this plot for American Airlines.

As figure 2.4 shows, American airlines experiences the highest median delay from late aircraft, and experiences the lowest median delay from late arrivals. The box-and-whisker plots offer an easy to see comparison between different variables, at the expense of the detail offered by histograms.

Next, we considered the correlations between the different variables of the dataset; we

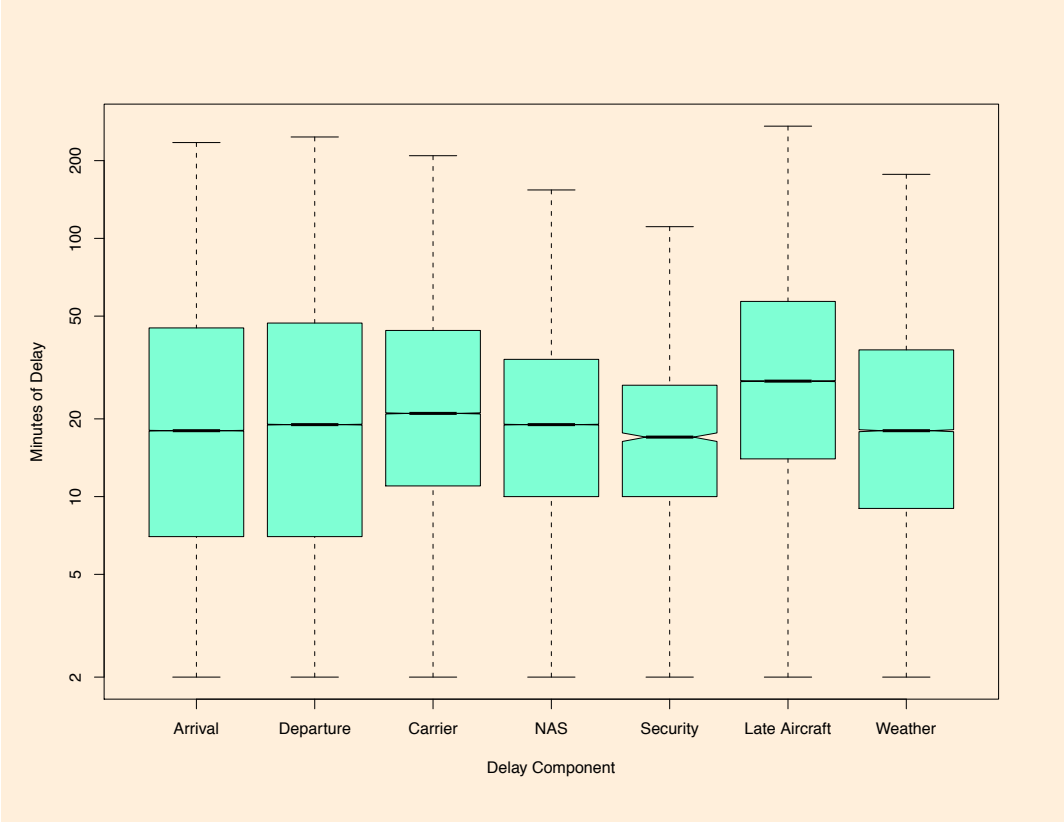


Figure 2.4: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

consider the year, arrival delay, arrival time, carrier delay, day of week, day of month, departure delay, departure time, distance, late aircraft delay, month, NAS delay, security delay, taxi in, taxi out, weather delay, and air time variables. We computed and plotted the correlation matrix of these variables for each carrier; figure 2.5 shows the results.

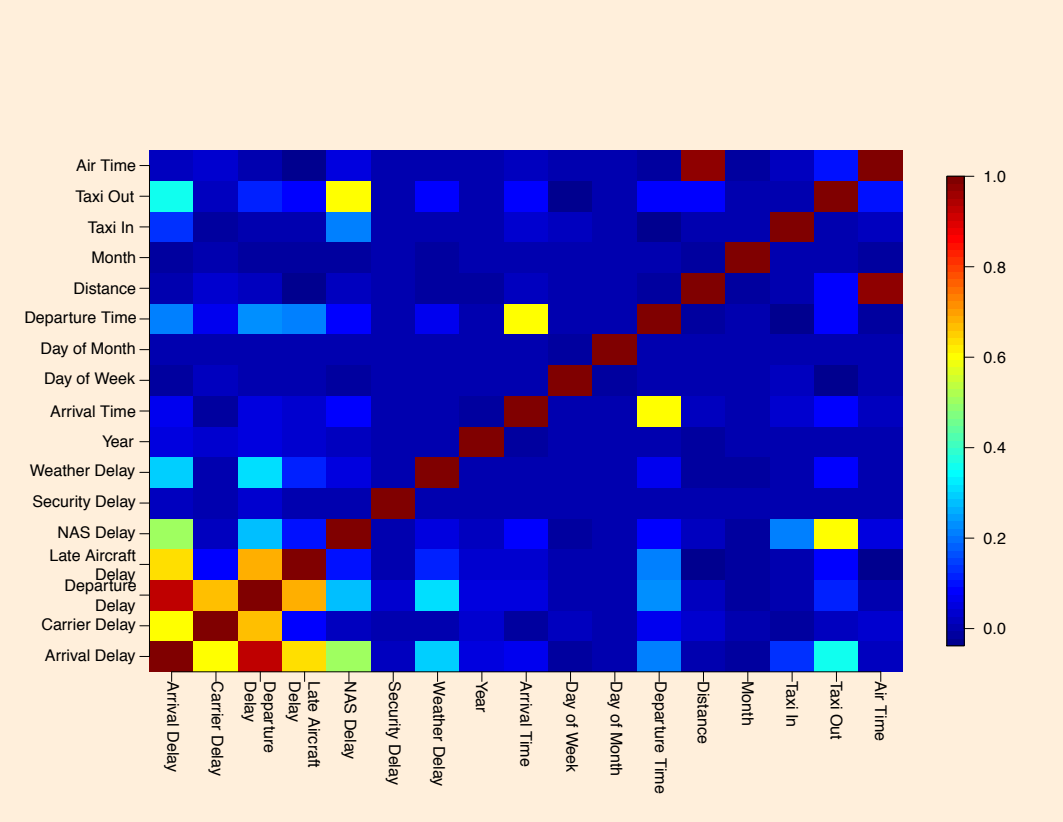


Figure 2.5: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

Note that the cells in the lower left corner of the correlation matrix tend to have a large positive correlation. These cells correspond to the correlation between the different components of delay; because of this, it is not surprising that these cells have a large correlation value, since it is to be expected that, for example, late aircraft delay tends to coincide with both arrival and departure delay. Another notable cell is the one corresponding to the distance and air time variables; as flights that cover a larger distance require a longer time in the air, it is no surprise that there is a nearly perfect positive correlation between the two

variables. A similarly obvious connection exists between the arrival time and departure time variables; as departure time increases, it is natural that arrival time might increase as well. Finally, we note that there is a somewhat smaller positive correlation between the taxi out time and the NAS delay variables. It could be expected that NAS delay increases taxi out time, as the delay may occur on the tarmac.

Regardless of possible explanations, these the correlations discussed above are clear outliers when compared to the remaining ones. However, this is not an indication that the other variables have no effect on each other. To determine which variables are significant indicators of flight delay, we must employ additional techniques.

Chapter 3

Statistical Analysis

We wished to be able to predict the amount of delay on a given flight. In general, we are able to specify characteristics of the flight, such as the date, the carrier, and the origin and destination of the flight. To make predictions about future flights, we analyzed the behavior of previous flights with similar characteristics; thus, also available for inspection are the actual delays realized on the flight. In total, data was available for the following variables: the day of the week, day of the month, month, year, the arrival and departure times, the distance traveled, and the arrival, departure, carrier, late aircraft, NAS, security, taxi in, taxi out, and weather delay times of the flight. We refer to these as *predictor variables*, which we relate in some way to our *response variable*, delay time. Our goal was to model this relationship.

3.1 Linear Regression

Possibly the simplest relationship between two variables is a linear one; given some predictor variable X and a response variable Y, a linear relationship between the two can be described as

$$Y = \beta_0 + X\beta_1 + \varepsilon,$$

where ε is a random error term.

Suppose that we are attempting to model a data set with observed indicator values $\mathbf{X} \in \mathbb{R}^m$ and response variables $\mathbf{Y} \in \mathbb{R}^m$. Our goal is to determine β_0 and β_1 such that

$$\hat{\mathbf{Y}} = \beta_0 + \mathbf{X}\beta_1$$

is a good estimator of the observations $\mathbf{Y}$.

We now generalize our model to include n indicator variables. We define the matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ as having entries X_{ij} corresponding to the i^{th} observation of the j^{th} indicator variable, $\mathbf{Y} \in \mathbb{R}^m$ as the vector of observed response variables, and $\boldsymbol{\beta} \in \mathbb{R}^n$ as the vector of weights on each indicator variable. Then the linear regression model can be generalized to the *multiple linear regression* model

$$\mathbf{Y} = \beta_0 + \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{3.1}$$

where $\boldsymbol{\varepsilon} \in \mathbb{R}^m$ is a vector of random error terms.

3.2 Least Squares Estimation

We now consider the question of how to discover appropriate values of $\boldsymbol{\beta}$ and β_0. Suppose we have a set of m observations of the indicator and response variables defining $\mathbf{X}$ and $\mathbf{Y}$. We wish to find $\boldsymbol{\beta}$ and β_0 such that the difference between the observed response and the predicted response are minimized; for example, we can consider the problem of minimizing the sum of the absolute value of the differences,

$$\min_{\beta, \beta_0} \|\mathbf{Y}_i - \beta_0 - \boldsymbol{\beta} X_i\|_1.$$

However, more accurate predictions are often found by minimizing the square of the differences, giving us the problem

$$\min_{\beta,\beta_0} \|\mathbf{Y} - \beta_0 - \mathbf{X}\boldsymbol{\beta}\|_2^2. \tag{3.2}$$

We define $\beta_0 = 0$, as we expect for a normal flight there will be zero flight delay. Then this problem has a closed form solution ([**?**]),

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{Y}. \tag{3.3}$$

Predictions using the $\hat{\boldsymbol{\beta}}$ estimator may have some amount of error; that is, there may be a difference between the observed value Y_i of the response variable and its predicted value, $\hat{Y}_i = \sum_{j=1}^{n} X_{ij}\hat{\beta}_i$. We measured this error with the *prediction error* of our model, given by

$$\text{PE} = \frac{1}{m}\sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2. \tag{3.4}$$

In order to encapsulate the type and magnitude of delay, we made use of a categorical response variable *DelayLevel* which is based on the sum of all delays. Figure 3.1 shows the delay values associated with each value of *DelayLevel*.

DelayLevel	0	1	2	3	4
Total amount of delay (minutes)	0 - 15	15 - 30	30 - 60	60 - 120	120+

Figure 3.1: The amount of delay associated with each value of the response variable *DelayLevel*.

To solve the least squares problem, we bootstrapped 70% of the data without replacement, and used this sample to generate an estimate of $\hat{\boldsymbol{\beta}}$. We repeated this process a large number of times, and used the mean $\hat{\boldsymbol{\beta}}$ estimator. Our motivation for bootstrapping the data was twofold. The first reason is a practical one; given the large size of our dataset, it is computationally expensive to perform a full multiple linear regression on the entire data

set. Second, we can use the remaining 30% of the data to form an estimate of our prediction error to validate our model. Figure 3.2 shows the prediction error when generating the β estimator for American Airlines over 100 independent bootstrap runs.

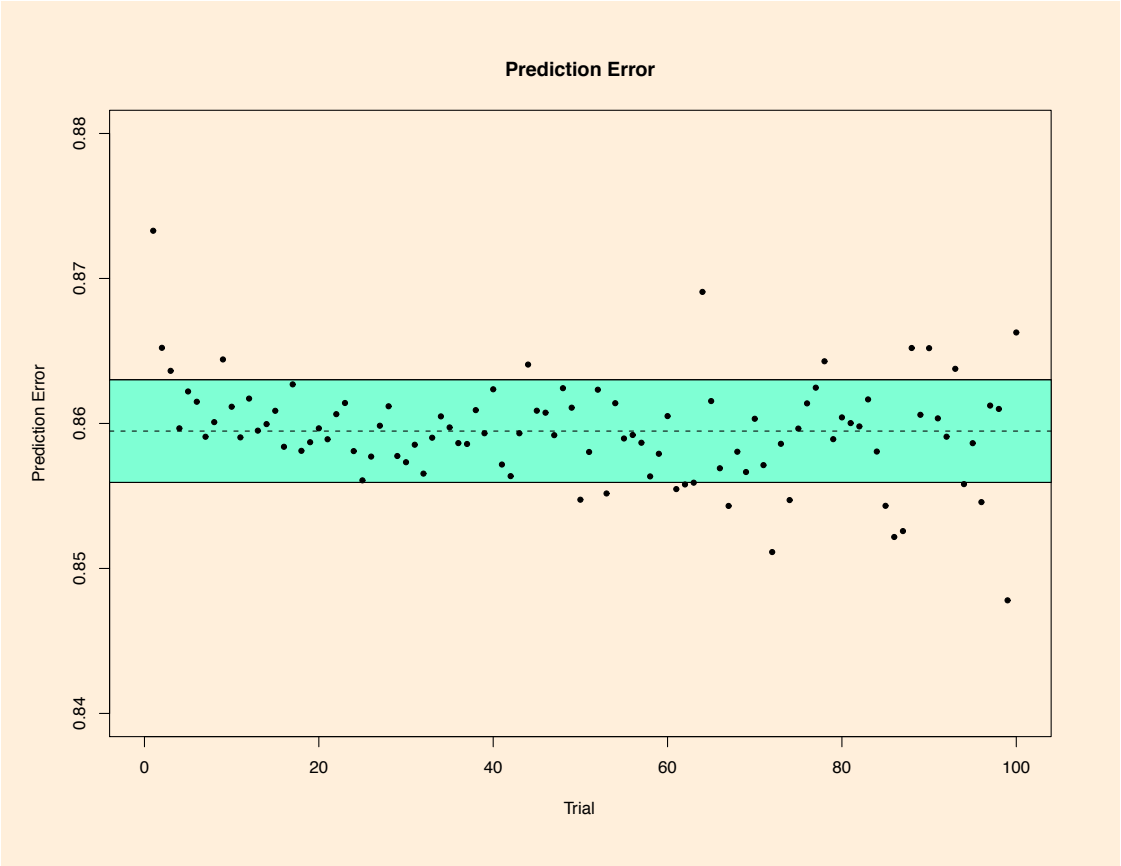


Figure 3.2: This plot shows the prediction error of our regression coefficients over 100 trials for the American Airlines data. Each point represents the calculated prediction error. The solid lines bounding the shaded region represent the region within one standard deviation of the mean, indicated by the dashed line.

Figure 3.3 shows the results of the multiple linear regression for American Airlines over the years 2005 to 2008.

American Airlines	$\hat{\beta}_i$
Year	4.3945e-02
Arrival Delay	6.5614e-02
Arrival Time	1.3257e-04
Carrier Delay	-6.9956e-02
Day of Week	4.4509e-03
Day of Month	6.1634e-04
Departure Delay	2.1746e-02
Departure Time	4.2327e-04
Distance	8.6416e-05
Late Aircraft Delay	-6.0670e-02
Month	-2.2380e-03
NAS Delay	-5.5275e-02
Security Delay	-4.9957e-02
Taxi In	3.4368e-02
Taxi Out	2.9167e-02
Weather Delay	-6.6875e-02

Figure 3.3: The $\boldsymbol{\beta}$ estimators found for American Airlines. The data for the other six carriers is shown in appendix B.

Chapter 4

Stochastic Optimization

Our goal is to solve the stochastic optimization problem. This chapter details our simulation approach towards solving the stochastic optimization problem, equation (1.2).

The objective of our simulation is to predict the shortest path through a network with respect to delay. To accomplish this, we specified the parameters that define the network in question, such as the day of the week, and day of the month of the flight, as well as the carrier. We then used historical data to populate a network with flights matching these parameters, and by some means computed the weights of each arc; we discuss these techniques in section 4.3. Finally, we solve the shortest paths problem from some specified origin. Algorithm 6 below demonstrates our method; the subroutines `reduce-network` and `sample-network` are described in sections 4.1 and 4.2 below, respectively.

4.1 Network Reduction

The first step in our simulation is to construct the network that we wish to simulate. We began with the network shown in figure 4.1, which is the set of all commercial, domestic flights from 2005 to 2008, which includes approximately 120 million distinct flights. However, simulating the delay in this network is computationally infeasible, and so we attempted to reduce its size.

Algorithm 6 This algorithm generates a network to solve the shortest paths problem utilizing historical data that represents the target network, and then solves the shortest paths problem.

Define: `simulation`
Require: *networkParameters*, *origin*, *destination*, *maxArcs*
  graph $\mathcal{G} \leftarrow$ `reduce-network`(*origin*, *destination*, *maxArcs*)
  for *i=0* **to** *NUM_SAMPLES* **do**
    $\mathcal{G}_i \leftarrow$ `sample-network`($\mathcal{G}$, *networkParameters*)
    $(\pi, d) \leftarrow$ `dijkstra`($\mathcal{G}$, *origin*)
    $(paths_i, weights_i) \leftarrow$ `backsolve`(π, d)
  end for
  return $(paths, weights)$

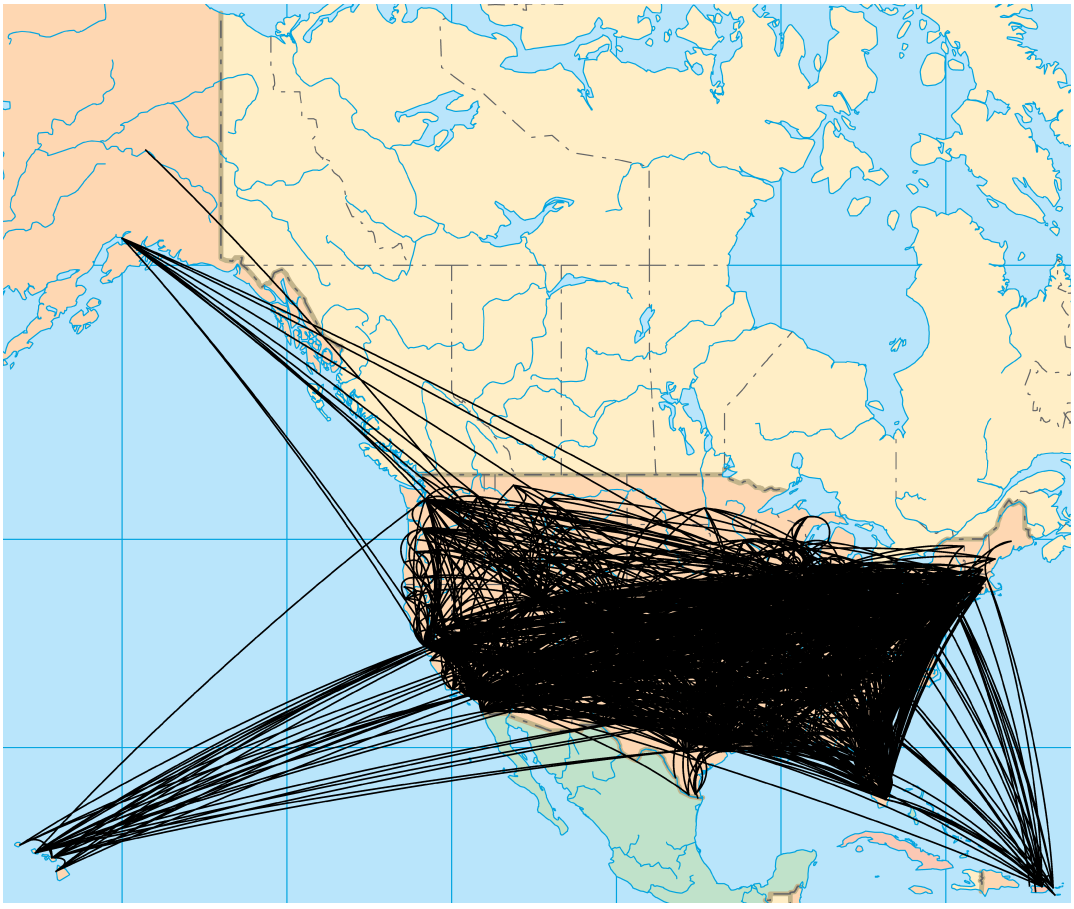


Figure 4.1: All flights flown by the major carriers considered over the 2005 to 2008 period.

We first considered a practical limitation on the network; when considering a flight path between two airports, travelers generally prefer flights with the fewest number of legs. Each additional flight leg introduces the possibility of baggage loss and flight delay, while also possibly requiring lengthy layovers. Motivated by this, we considered reducing the network to include only arcs that can be reached within k flights, where k is generally chosen to be two or three. This is accomplished by applying the breadth-first search techniques discussed in section 1.3.2. We perform a breadth-first search from our origin airport, and remove any nodes found to be more than k arcs from it. We then included all arcs from the original network that are between the remaining nodes. Figure 4.2 shows the result of this technique on the network in full network from figure 4.1 using an origin airport of Boston Logan International.

The network in figure 4.2 contains just under 2000 arcs, while the full network contained over 3300. While this is a substantial reduction in size, it comes from eliminating one direction of many arcs, and so does not significantly alter the structure of the graph. To improve upon this method, we imposed an additional restriction on the arcs included in our reduced network. Previously, after we selected which nodes to include in the network, we then included all arcs between these nodes present in the network. Using the same set of nodes, we only included arcs in the network that are part of a k-path from our origin node to our destination node. This results in the smallest network that can be considered that contains all k-paths from the origin node to the destination node. Figure 4.3 demonstrates the effects of this technique when using Boston Logan International as the origin airport and Los Angeles International as the destination airport.

As figure 4.3 shows, the reduced network contains only 58 arcs, compared to the 3367 present in the original flight network. However, if we choose $k = 3$, then the network expands to 1430 arcs.

To accomplish this reduction, we needed to be able to identify all the k-paths in a network. The set of k-paths could be found by enumerating the paths in the network;

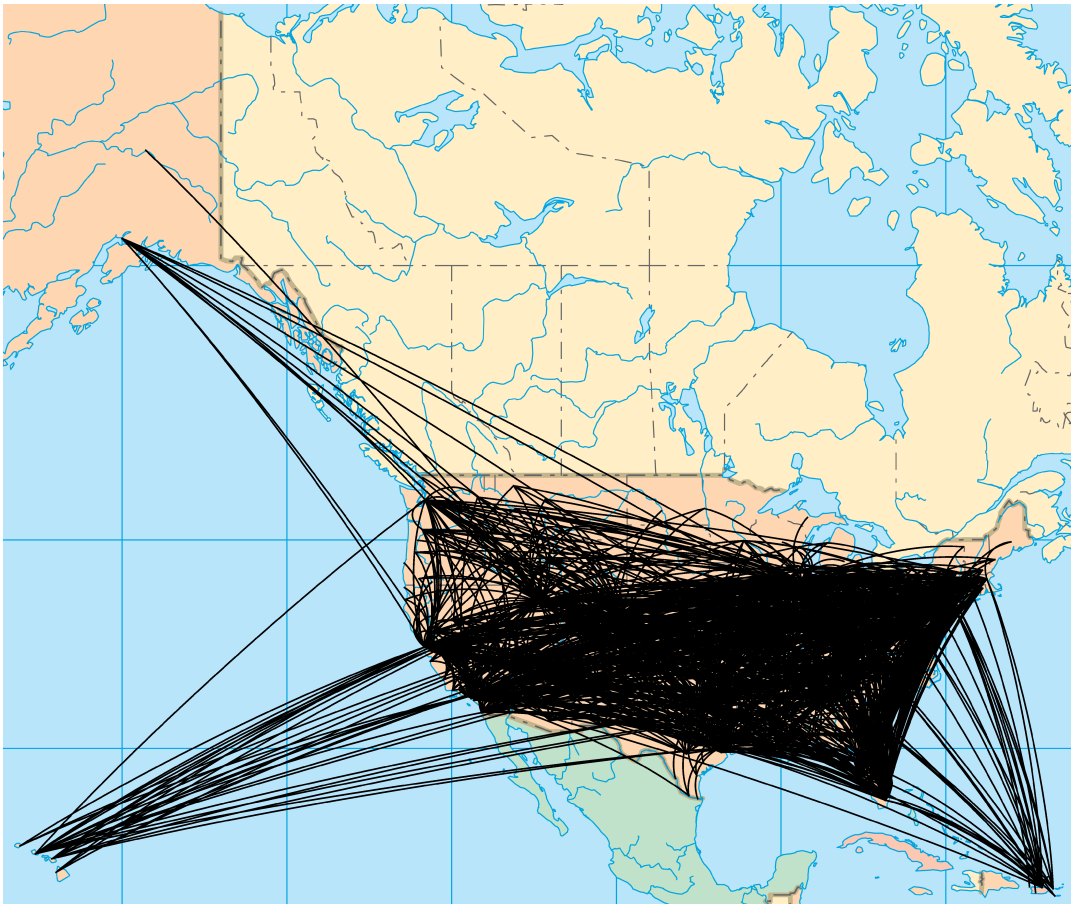


Figure 4.2: All arcs that can be traversed within two arcs of Boston Logan International. Note that while there appears to be no significant difference between this network and the full network in figure 4.1, this network in fact contains just under 2000 arcs, while the previous network contained over 3300. This reduction comes from eliminating one direction of each bi-directional arc.

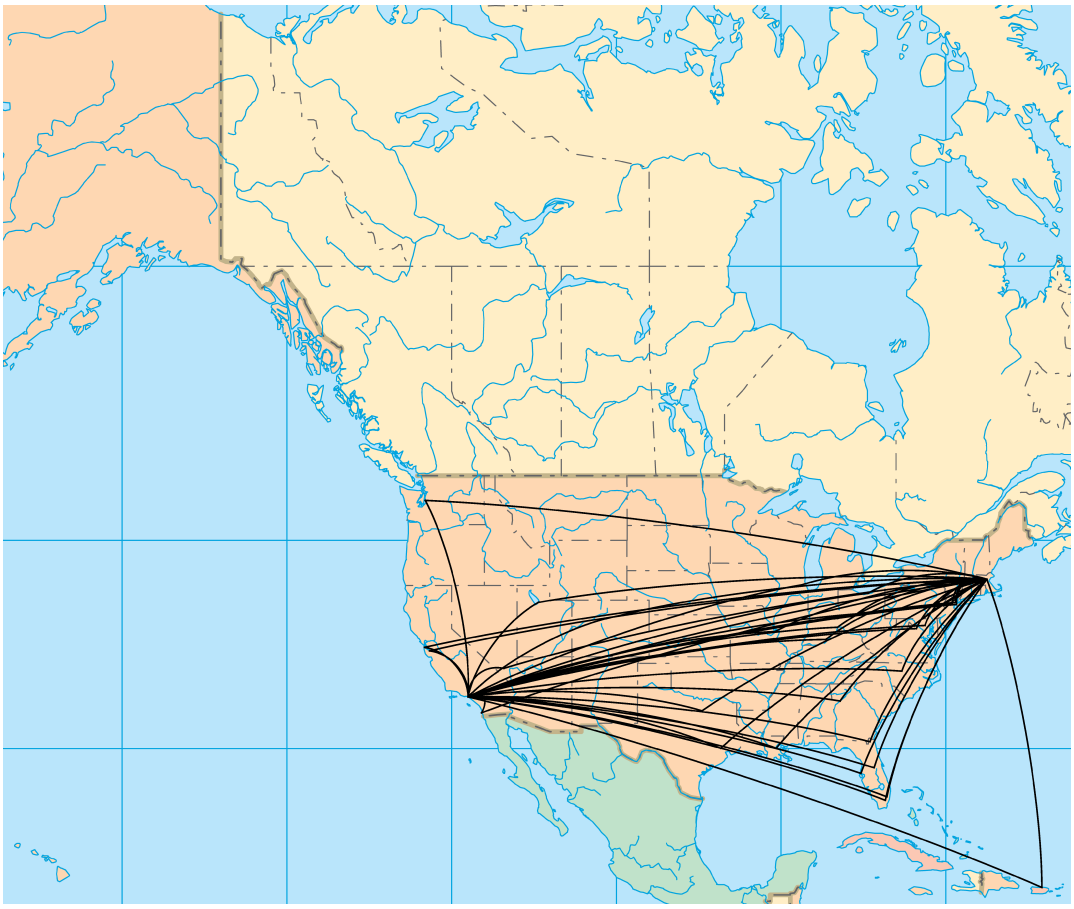


Figure 4.3: The network containing only k-paths from the origin to the destination; in this case, $k = 2$, and the origin is Boston Logan International, and the destination in Los Angeles International.

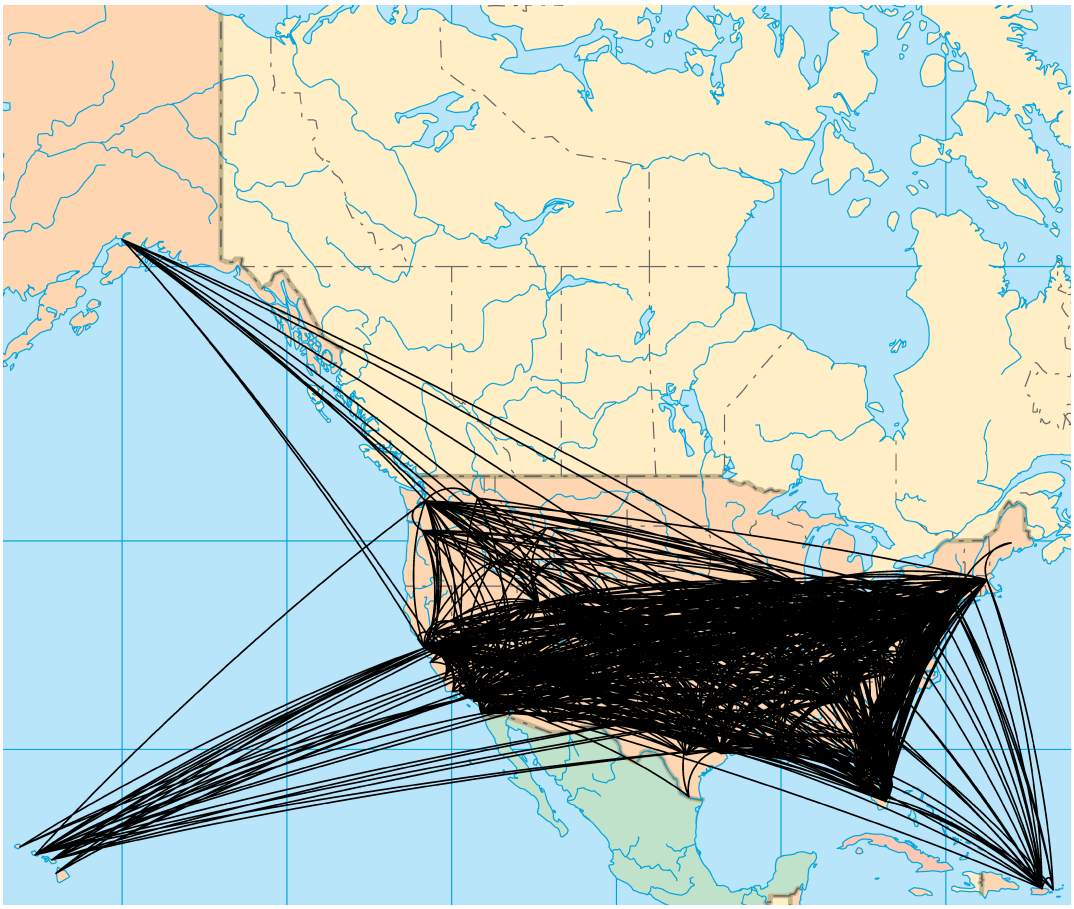


Figure 4.4: The fully reduced flight network, choosing $k = 3$, with Boston Logan International and Los Angeles International as the origin and destination airports, respectively.

however, the number of paths in the network is quite large. To overcome this, we identified conditions for a given arc to be included in a k-path. Lemma 1 provides us with such a condition.

Lemma 1. *Allow a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, nodes $a, b \in \mathcal{V}$, and $k \geq 1 \in \mathbb{Z}$ be given. Define the function D from $\mathcal{V} \times \mathcal{V} \to \mathbb{Z}^+$ determine the fewest number of arcs between a and b in $\mathcal{X}$; that is,*

$$D_{\mathcal{G}}(a,b) = \begin{cases} \min\{k : \exists\, a \xrightarrow{k} b\}, & \exists\, a \to b, \\ \infty, & \textit{otherwise} \end{cases}$$

Consider the reversed graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$, where

$$\mathcal{E}' = \{(a,b) \in \mathcal{V} \times \mathcal{V} : (b,a) \in \mathcal{E}\}.$$

Then the graph $\mathcal{F} = (\mathcal{V}_{\mathcal{F}}, \mathcal{E}_{\mathcal{F}})$, where

$$\mathcal{E}_{\mathcal{F}} = \{(s,t) \in \mathcal{V} \times \mathcal{V} : D_{\mathcal{G}}(a,s) + D_{\mathcal{G}'}(b,t) < k\}$$

$$\mathcal{V}_{\mathcal{F}} = \{a \in \mathcal{V} : (a,b) \in \mathcal{E}_{\mathcal{F}}\} \cup \{b \in \mathcal{V} : (a,b) \in \mathcal{E}_{\mathcal{F}}\}$$

is the set of all k-paths from a to b in $\mathcal{G}$. Additionally, $\mathcal{F}$ contains only arcs that are part of a k-path from a to b in cG.

Proof. We first show that if there exists a $a \xrightarrow{k} b$ path in $\mathcal{G}$, then it is also in $\mathcal{F}$. Allow our origin node to be a_1 and our destination node to be a_{k+1}, and allow a k-path $(a_1, ..., a_{j+1})$ to be given, where $1 \leq j \leq k$. Let an arc (a_i, a_{i+1}) in this k-path be given, and allow the reversed graph $\mathcal{G}'$ be defined as before. Note that $D_{\mathcal{G}}(a_1, a_i) \leq i-1$, since the first i nodes in our k-path form an $(i-1)$-path $(a_1, ..., a_{i-1})$; similarly, we have that $D_{\mathcal{G}'}(a_{j+1}, a_{i+1}) \leq k-i$, since the existence of the $a_1 \xrightarrow{k} a_{k+1}$ path ensures the existence of the $(j-i)$-path

$(a_{j+1},...,a_{i+1})$. Thus, for any arc (a_i,a_{i+1}) in a k-path from a_1 to a_{j+1}, we have that

$$\begin{aligned} D_{\mathcal{G}}(a_1,a_i)+1+D_{\mathcal{G}'}(a_{j+1},a_{i+1}) &\leq (i-1)+1+(j-i) \\ &\leq j+1 \\ &\leq k, \end{aligned}$$

and so the graph $\mathcal{F}$ contains each arc in the k-path, as required.

We now show that if an arc is in $\mathcal{F}$, then it is part of an $a \xrightarrow{k} b$ path in $\mathcal{G}$. Allow the arc $(c,d) \in \mathcal{F}$ be given. We have by definition that $D_{\mathcal{G}}(a,c)+D_{\mathcal{G}}(d,s) < k$. Then we have that there exist two paths $a \to c$ and $d \to b$ of total length less than k in $\mathcal{G}$; thus, by connecting these two paths with the arc (c,d), a k-path from a to b in $\mathcal{G}$ is formed, and so (c,d) is part of a $a \xrightarrow{k} b$ path in $\mathcal{G}$, as required. □

To utilize this lemma, we must know not only the minimum number of arcs between the origin node and all other nodes in the network, but also from the destination node and all other nodes in the revered network. As discussed in section 1.3.2, this can be accomplished with only two breadth-first searches – one on the original network, and one on the reversed network.

4.2 Sampling

To solve the shortest paths problem, the flight network must first be simulated. We utilized two simulation methods, a naïve method, and a more sophisticated method that accounts for the time-dependent nature of the network. In both cases, the goal is to assign weights to the network created via our network reduction techniques. We refer to the first method as the naïve method, and the improved method the cascade method. We discuss method each in turn.

层叠

In this section, when we refer to 'the network', we are referring to any network graph; we make no assumptions about the structure of the graph. Additionally, we assume that

Algorithm 7 This algorithm reduces the network to the set of k-paths from the specified origin to the specified destination.

Define: `reduce-network`
Require: $origin, destination, k$
  $\mathcal{G} = (\mathcal{V}, \mathcal{E}) \leftarrow$ U.S. Flight Network
  $\mathcal{G}' = (\mathcal{V}', \mathcal{E}') \leftarrow$ Reversed U.S. Flight Network
  $(\pi, d) \leftarrow \texttt{BFS}(\mathcal{G}, origin)$
  $(\pi', d') \leftarrow \texttt{BFS}(\mathcal{G}', destination)$
  $\mathcal{F} = (\mathcal{V}_\mathcal{F}, \mathcal{E}_\mathcal{F}) \leftarrow (\emptyset, \emptyset)$
  for node $n \in \mathcal{V}$ **do**
    for node $m \in \mathcal{V}$ **do**
      if $\pi(n) + \pi'(m) < k$ **then**
        $\mathcal{V}_\mathcal{F} \leftarrow n$
        $\mathcal{V}_\mathcal{F} \leftarrow m$
        $\mathcal{W}_\mathcal{F} \leftarrow (n, m)$
      end if
    end for
  end for
  return $\mathcal{F}$

we have parsed our data set to find all flights that have the requisite characteristics for our network; namely, this data set contains all flight records from arcs in the network that match the problem parameters given, such as day, week, or month of travel, and the carrier.

4.2.1 Naïve Sampling

The naïve method utilizes independent sampling to populate the network. Algorithm 8 shows the psuedocode describing this method. For each arc in our chosen network, we randomly and uniformly select a flight record from the data set corresponding to that arc. We then compute the arc's weight by one of the methods discussed in 4.3, using this flight record as input.

We initially utilized this method because of its simplicity; it is easy to compute the arc weights using this method, as it only requires one sample per arc, with very little other computation besides memory operations. However, this method has a number of flaws, namely,

Algorithm 8 Assigns weights to a network graph $\mathcal{G}$ by uniformly sampling from a data set *data* for relevant flight records. We define *data set*[*flight*] as the subset of flight records corresponding to the flight *flight*. The algorithm `compute-weight` can be implemented as any of the methods discussed in section 4.3.

Define: `naive`
Require: $\mathcal{G}$, *data*
  for *arc* **in** $\mathcal{G}$ **do**
    record $\leftarrow$ uniform random sample from *data*[*arc*]
    arc.weight $\leftarrow$ `compute-weight`(*record*)
  end for
  return $\mathcal{G}$

1. Flights in the network are not independent of each other; the same plane may be used for multiple flights in the same day, and so a delay affecting one flight can certainly affect another.

2. Independent sampling ignores time-dependent nature of the network; delay may tend to increase or decrease throughout the day.

3. Sampled arcs may not constitute a realizable network; departing flights may be chosen that depart before arriving flights.

To overcome these limitations, we turned to cascade sampling.

4.2.2 Cascade Sampling

Cascade sampling attempts to address the time-dependent nature of the flight network graphs. Algorithm 9 describes this method. To accomplish this, we ensured that any sampled flight departs after the previous flight in the network has arrived. To enforce this condition, we first sampled all flights leaving the origin airport. We then sampled all flights leaving the destinations of these flights, ensuring that we only sample from flights departing after they arrive. We continued this process until all arcs in the graph have been assigned a weight.

Algorithm 9 Assigns weights to a network graph $\mathcal{G}$ by sampling from a data set *data* for relevant flight records; unlike `naive`, this algorithm takes into account the time-dependence of the network. We define *data*[*flight*,*t*] as the subset of flight records corresponding to the flight *flight* that depart after time t on a given day. An initial call to `cascade` would likely set $t = 0$, while recursive calls to `cascade` will set t to later times. The algorithm `compute-weight` can be implemented as any of the methods discussed in section 4.3.

Define: `cascade`
Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, *origin*, start time t
  for $b \in \mathcal{V}$ s.t. $(origin, b) \in \mathcal{E}$ **do**
    $arc \leftarrow (origin, b)$
    $record \leftarrow$ uniform random sample from data after time t
    $\mathcal{W}_{arc} \leftarrow$ `computeWeight`($record$)
    $\mathcal{G} \leftarrow$ `cascade`$((\mathcal{V}, \mathcal{E}, \mathcal{W}), data, arc, record.arrivalTime)$
  end for
  return $\mathcal{G}$

4.3 Arc Weights

As of yet, we have not considered how to weight the arcs in the network. We have considerable flexibility in choosing how to do this, since all other components of the algorithm are independent of how the weights are chosen; all they require is that some weight is chosen.

We considered several possible weighting schemes; each scheme was used to minimize a different objective. In section 4.3.1, we outline a technique that uses the observed, rather than simulated, delay along a route as arc weights. In section 4.3.2, we demonstrate a technique that uses the weighted sum of different variables of a flight for its weight. These first two methods are suitable for minimizing flight delay. In section 4.3.3, we outline the technique of using the total time of a flight as its weight. This method is suitable for minimizing the total travel time between the origin and destination. Finally, in section 4.3.4, we demonstrate a technique that uses a weighted combination of the delay and total time of a flight as its weight. This technique is useful for mixed-objective minimization problems, where the user wishes to minimize both delay and the total travel time.

Algorithms 8 and 9 utilize the algorithm `computeWeight`, which took as an argument one flight record; each of the following weighting schemes assigns a weight to an

arc based on the values in a flight record, and so can be considered the return values of `computeWeight`; which one will actually be used is specified at run time.

4.3.1 Historical Delay Weighting

The first weighting scheme we considered uses the actual historical delay observed on a given flight as the arc weight. That is, for any flight record, the corresponding weight is simply the sum of all the delay variables. If our delay variables are $d_i, i = 1, ..., m$, then the weight w is given by

$$w = \sum_{i=1}^{m} d_i. \tag{4.1}$$

4.3.2 Component Delay Weighting

The next weighting scheme we considered utilizes the categorical delay variables computed in chapter 3. With this method, each arc is assigned a categorical delay value c between 0 and 5; if v_i and β_i, $i = 1, ..., n$ are the data set variables from the flight and associated β values, then c is defined as

$$c = \sum_{i=1}^{n} \beta_i v_i. \tag{4.2}$$

4.3.3 Total Time Weighting

The next weighting scheme is used to minimize the total travel time, rather than just delay. The weight assigned here is simply the time of flight given by a flight record; note that this time includes any amount of delay experienced, if any. If t_a is the air time of the flight, t_i is the taxi in time, and t_o is the taxi out time, then the weight assigned is simply

$$w = t_a + t_i + t_o. \tag{4.3}$$

4.3.4 Mixed Weighting

This weighting scheme weights an arc based both on total travel time and by the amount of delay experienced on that flight. Let t_1 be the total flight time, and t_2 be the total delay time; t_2 may be computed either by summing the historical delay values, or by utilizing the categorical delay weighting. Then the weight w assigned to the arc is

$$w = \theta_1 t_1 + \theta_2 t_2, \tag{4.4}$$

where θ_1, θ_2 are real, nonnegative values supplied by the user.

Chapter 5

Results and Findings

5.1 Minimum Delay Routes

We now explore the uses of the algorithms that we have developed. We considered finding the minimum delay routes between Boston Logan International (BOS) and Los Angeles International (LAX) airports. Figures 5.1, 5.2, 5.3, and 5.4 show the solutions to the shortest paths problem when solved using different techniques. Figure 5.1 uses the simplest approach, utilizing naïve sampling and observed delay values for simulation. Figure 5.2 also uses naïve sampling, but uses the regressed delay values computed via the techniques discussed in chapter 3. Figure 5.3 uses cascade sampling with observed delay values, and figure 5.4 uses cascade sampling with regressed delay values. In these graphs, the presence of an arc indicates that that flight path was chosen as the shortest path at least once during the simulation. The color of the arc indicates the average delay experienced along that route, and the width of the arc indicates the relative frequency with which a route was chosen as the shortest path. For example, blue arcs have low average delay compared to red arcs, and narrow arcs were chosen as the shortest path a fewer number of times than a thick arc.

Consider for a moment the difference between the solutions generated via naïve versus

cascade sampling. Overall, the solutions appear to be very similar; in fact, all four solution methods found the direct flight from BOS to LAX to be the shortest path the most number of times, as evidenced by the width of the arc. However, predictions made using the naïve sampling technique tend to have lower delays than those made with cascade sampling. In particular, predictions made with the cascade method demonstrate the propagation of delay from one flight to the next; to see this, note that almost every arc arriving at the destination in figures 5.3 and 5.4 has close to 60 minutes of delay, while the terminal arcs of figures 5.1 and 5.2 tend to have approximately the same delay as the previous flight on each path.

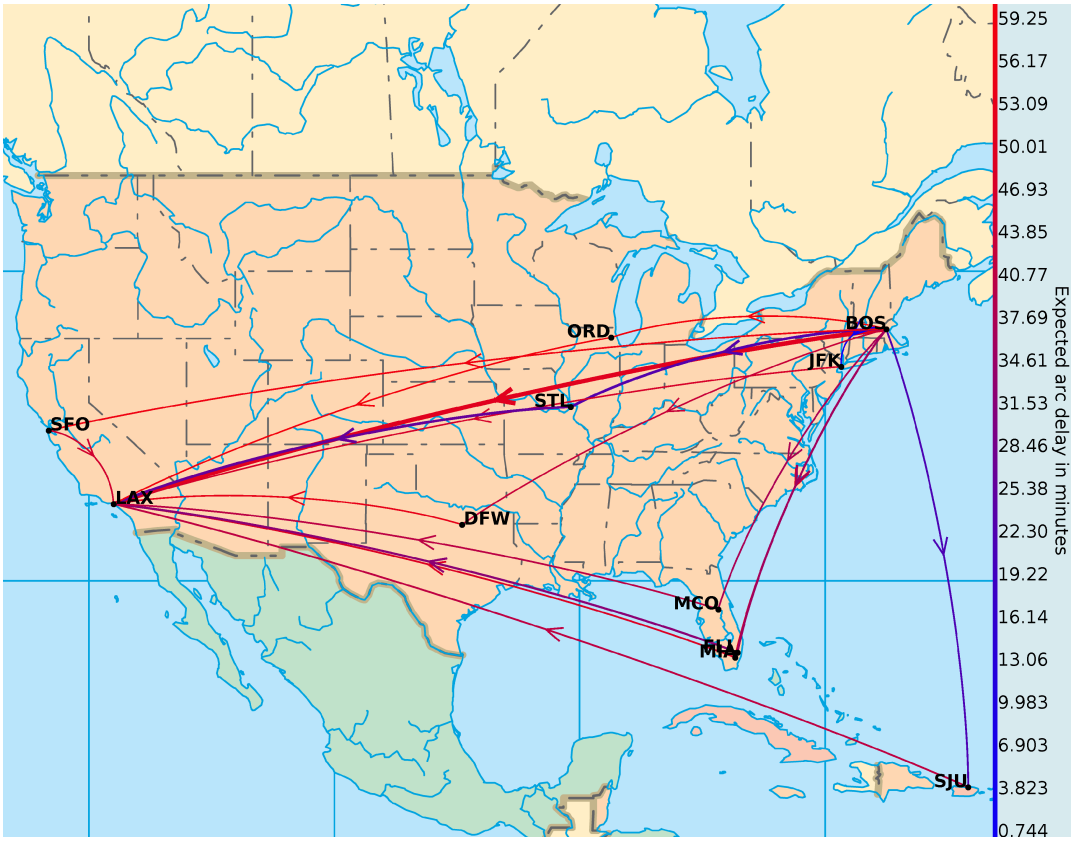


Figure 5.1: The solutions to the shortest paths problem from BOS to LAX for American Airlines flying on a Monday. The problem was solved using naïve sampling and the observed delay times. The presence of an arc indicates that that flight path was chosen as the shortest path at least once during the simulation. The color of the arc indicates the average delay experienced along that route, and the width of the arc indicates the relative frequency with which a route was chosen as the shortest path.

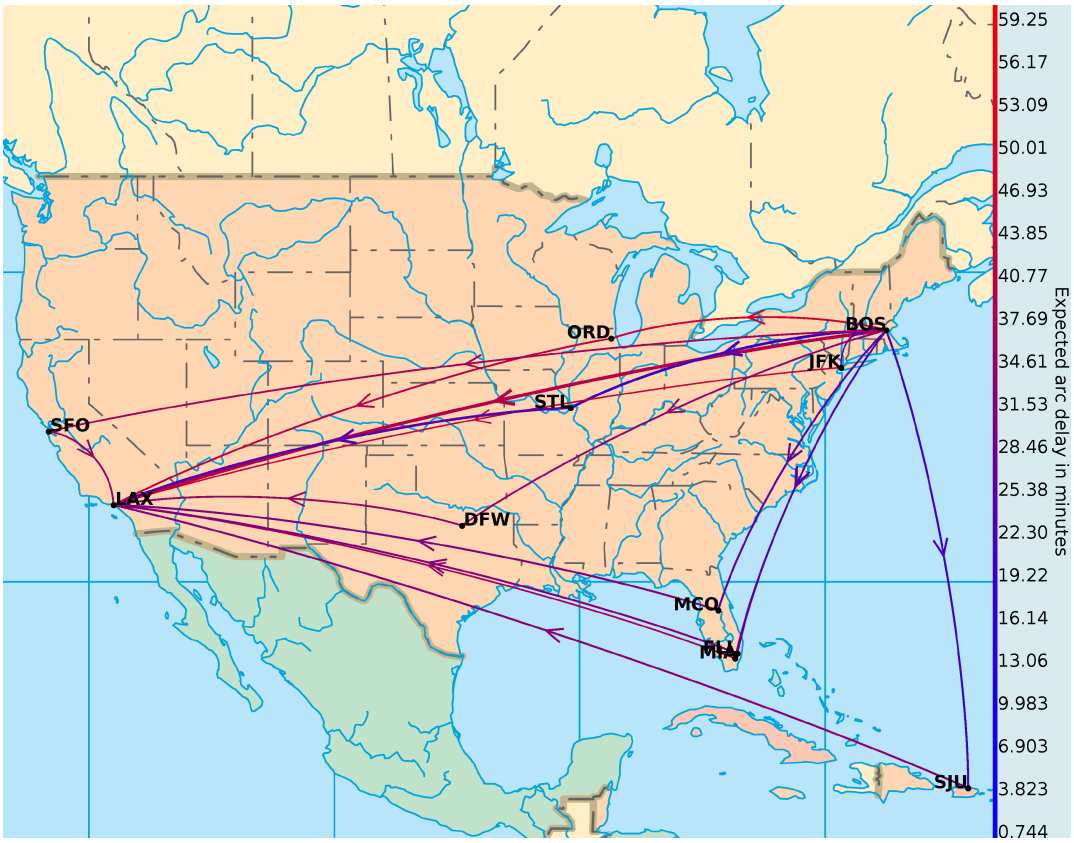


Figure 5.2: The solutions to the shortest paths problem from BOS to LAX for American Airlines flying on a Monday. The problem was solved using naïve sampling and the regressed delay times. The presence of an arc indicates that that flight path was chosen as the shortest path at least once during the simulation. The color of the arc indicates the average delay experienced along that route, and the width of the arc indicates the relative frequency with which a route was chosen as the shortest path.

Figures 5.5 and 5.7 show the solutions to the shortest paths problem when we allow flight paths with up to three arcs. Note that figure 5.5 has a only a few more arcs than the two-arc limited networks in figures 5.1 to 5.4, while figure 5.7 has quite a few more arcs than 5.6.

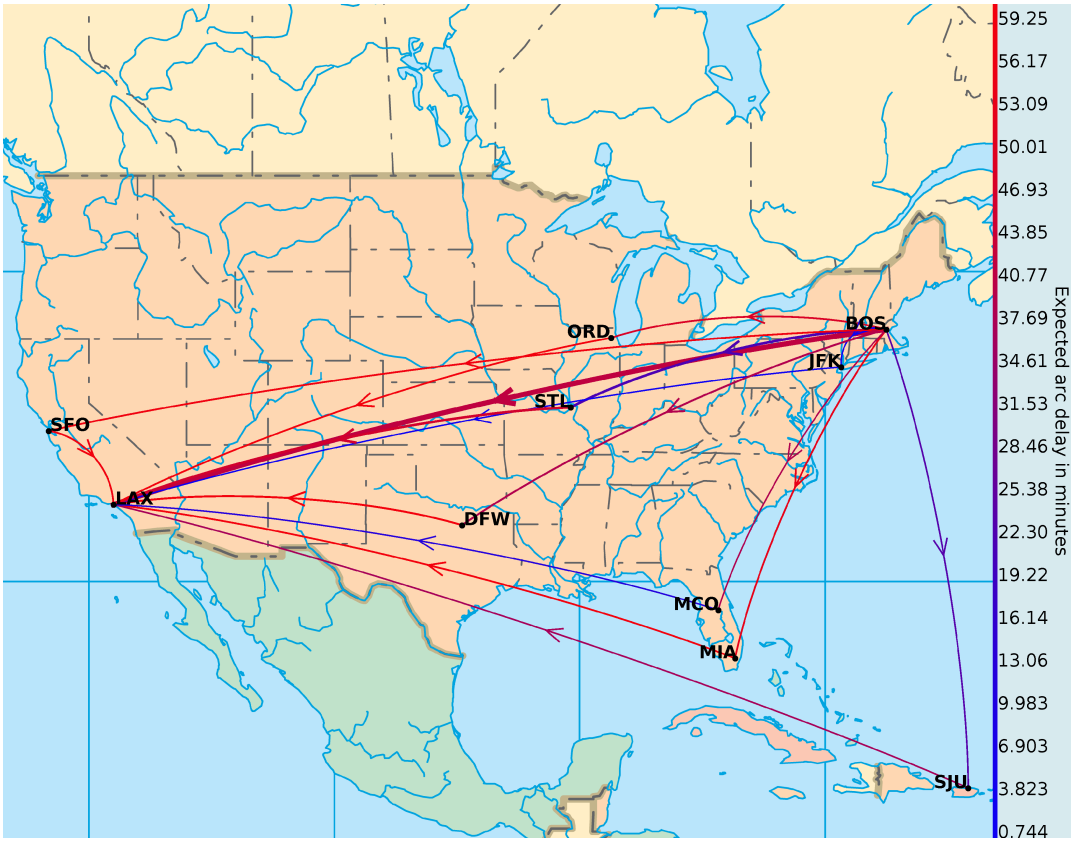


Figure 5.3: The solutions to the shortest paths problem from BOS to LAX for American Airlines flying on a Monday. The problem was solved using cascade sampling and the observed delay times. The presence of an arc indicates that that flight path was chosen as the shortest path at least once during the simulation. The color of the arc indicates the average delay experienced along that route, and the width of the arc indicates the relative frequency with which a route was chosen as the shortest path.

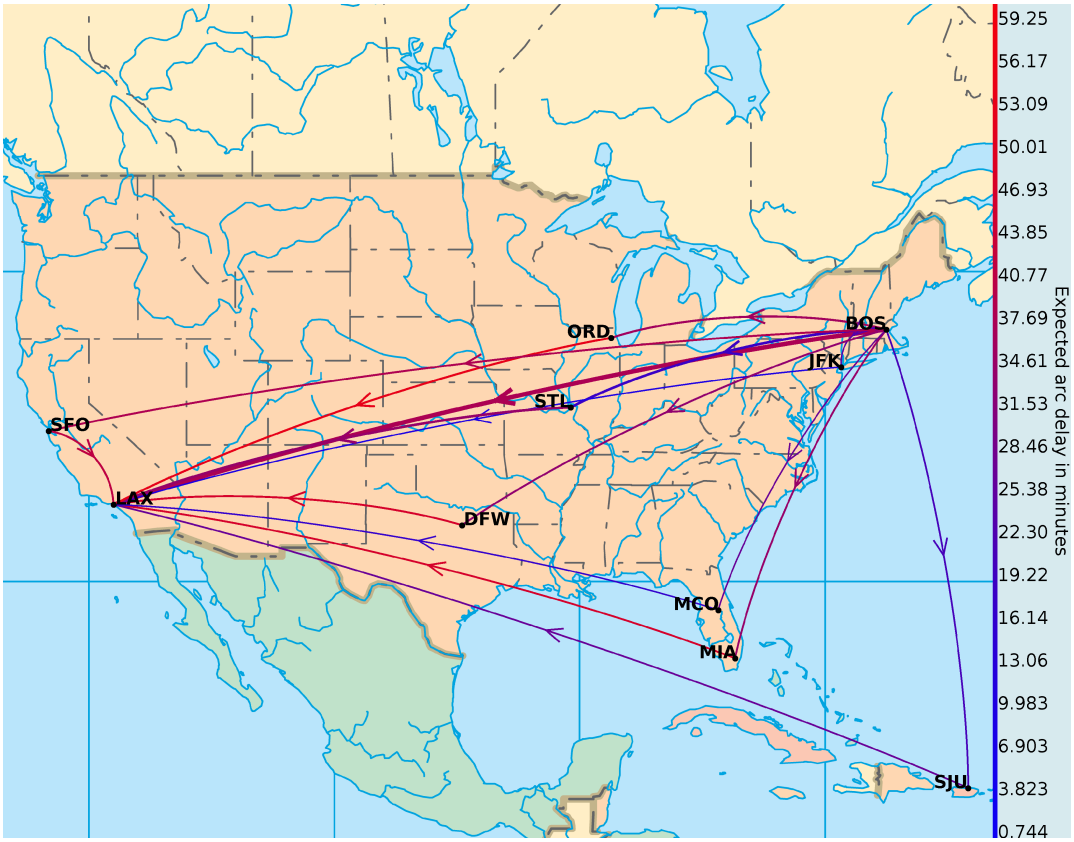


Figure 5.4: The solutions to the shortest paths problem from BOS to LAX for American Airlines flying on a Monday. The problem was solved using cascade sampling and the observed delay times. The presence of an arc indicates that that flight path was chosen as the shortest path at least once during the simulation. The color of the arc indicates the average delay experienced along that route, and the width of the arc indicates the relative frequency with which a route was chosen as the shortest path.

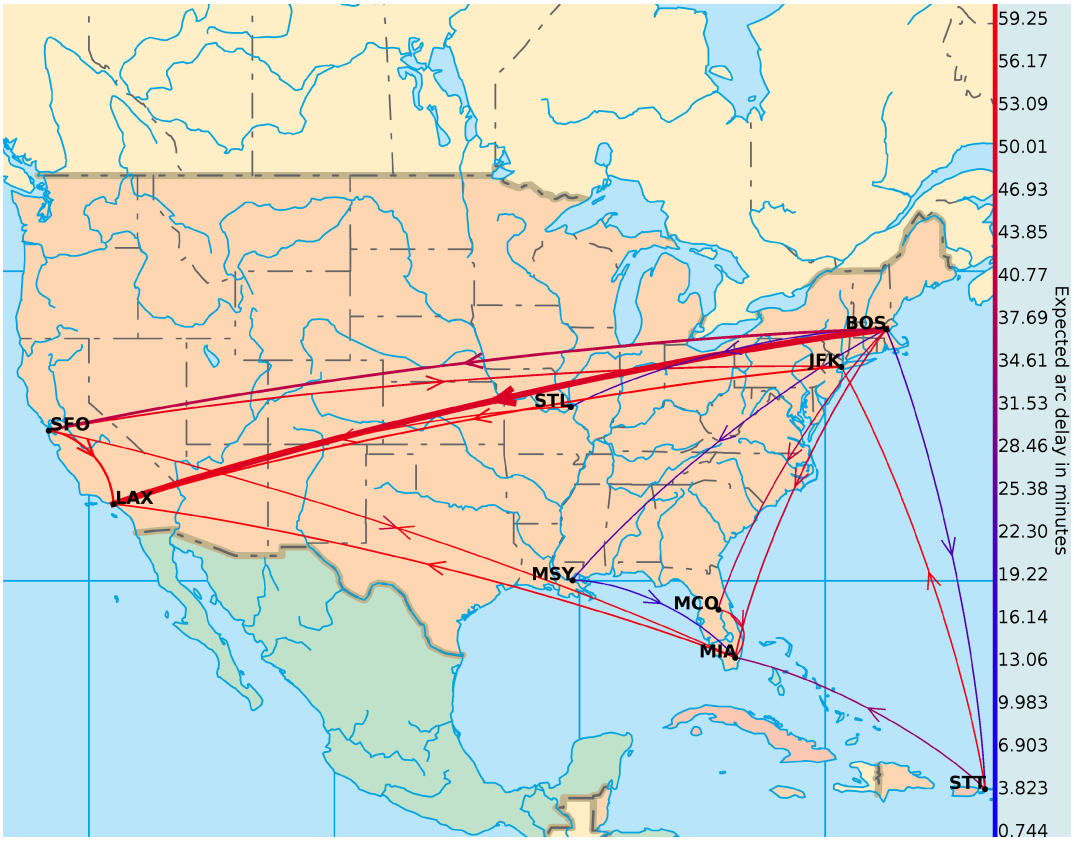


Figure 5.5: The solutions to the shortest paths problem from BOS to LAX for American Airlines, where we allow paths with up to three arcs. Note how this network is not substantially more complex than those in figures 5.1 to 5.4.

Figure 5.6: The solutions to the shortest paths problem from ORD to DFW for United Airlines, where we allow paths with up to two arcs.

Figure 5.7: The solutions to the shortest paths problem from ORD to DFW for United Airlines, where we allow paths with up to three arcs. Note how this network is substantially more complex than the network in figure 5.6

5.2 Visualization Tools for Networks

We can utilize our visualization tools to understand the behavior and structure of flight networks. Consider figures 5.8 and 5.9. In these graphs all flights are flown by Southwest airlines over various years. Each arc represents a single flight path, and the color of the arc indicates the average observed delay. Figure 5.8 represents data from the years 1987 and 1997, while figure 5.9 represents data from the years 2002 and 2008. As these graphs demonstrate, Southwest Airlines did, in fact, begin operating in the southwest United States, and expanded over the years. With this expansion, we can see a significant increase in the average delay along each route.

Now consider the arc-only figures 5.10 and 5.11. Like figures 5.8 and 5.9, these graphs show all flight paths flown by United Airlines over the years 1987, 1997, 2002, and 2008. These graphs suggest that, unlike Southwest's, the structure of United Airline's flight network has not changed significantly from 1987 to 2008; minor differences exist year to year, but overall the structure is quite constant. Like Southwest, these graphs also show a sharp increase in the average delay on most flights in the network in recent years. To develop a better understanding of United's flight network, we consider the volume of flights through each airport. Consider figure 5.12. In these graphs, each colored circle represents an airport. The size of the circle is proportional to the total number of flights to and from that airport, the color indicates the average observed delay given that there was more than 15 minutes of delay, and the opacity of the circle indicates the probability of encountering delay greater than 15 minutes. Note that in 1987 the number of flights at each airport was approximately the same, but by 2008, several airports clearly dominate the network; we refer to the airports with a comparatively large number of flights as 'hub' airports, while the smaller airports are 'spoke' airports.

We now analyze the Southwest network using this same type of graph; figure 5.13 shows the Southwest flight network in 1987 and 2008. As this figure shows, Southwest has also moved to more of a hub-and-spoke flight network; however, the difference in size

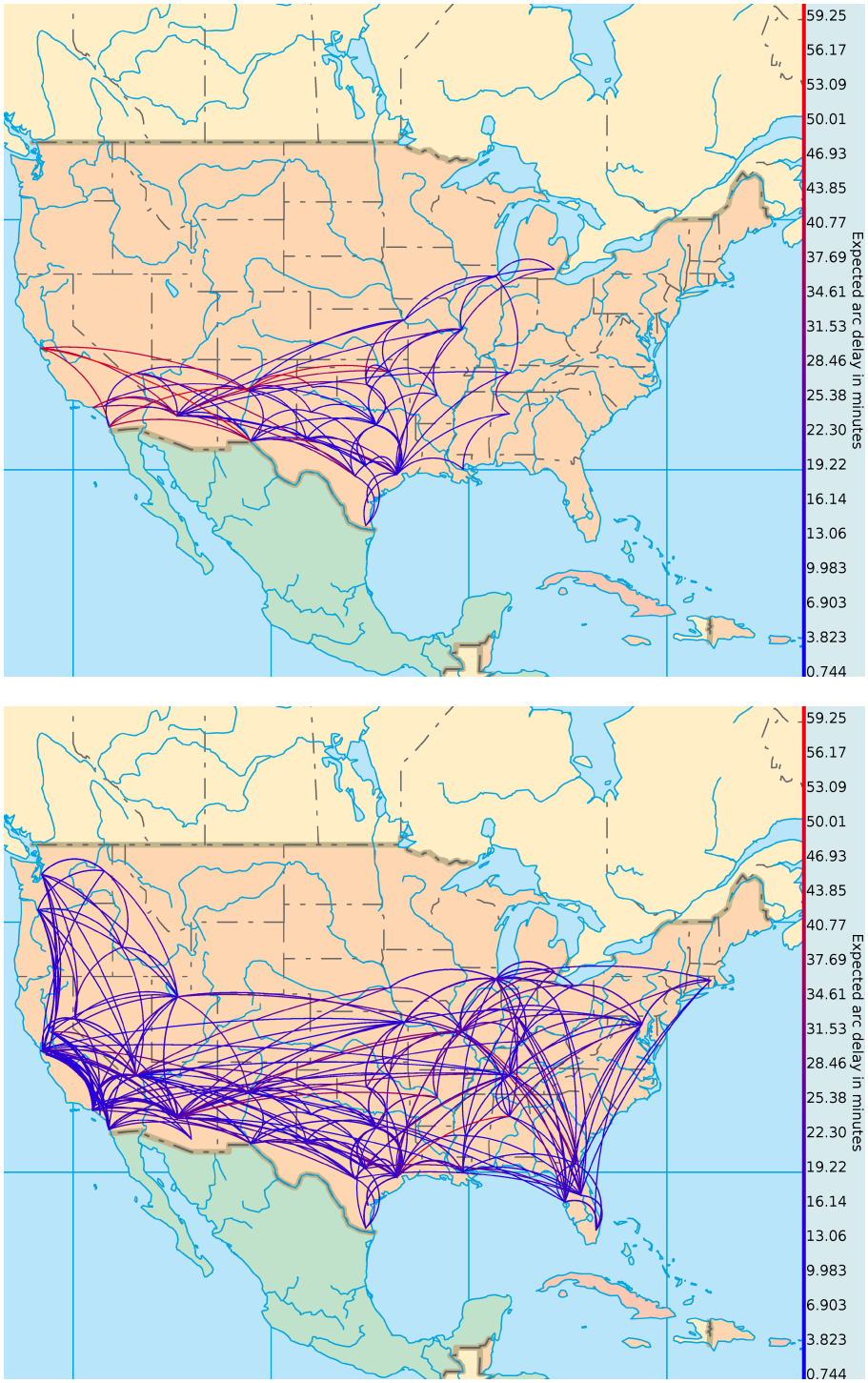


Figure 5.8: The Southwest flight network; the top graph shows the 1987 network, while the bottom shows the 1997 network. Arc color indicates average observed delay.

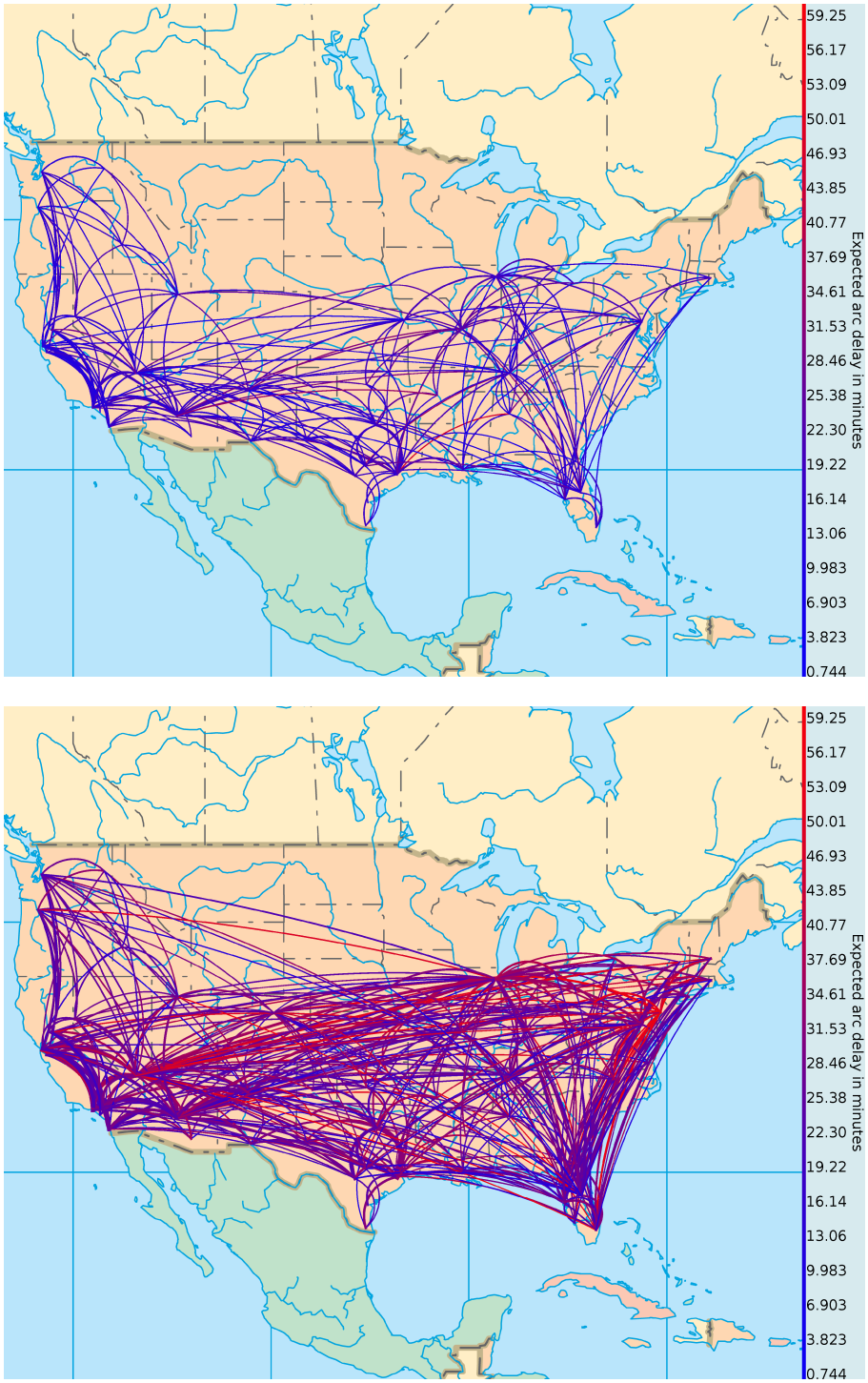


Figure 5.9: The Southwest flight network; the top graph shows the 2002 network, while the bottom shows the 2008 network. Arc color indicates average observed delay.

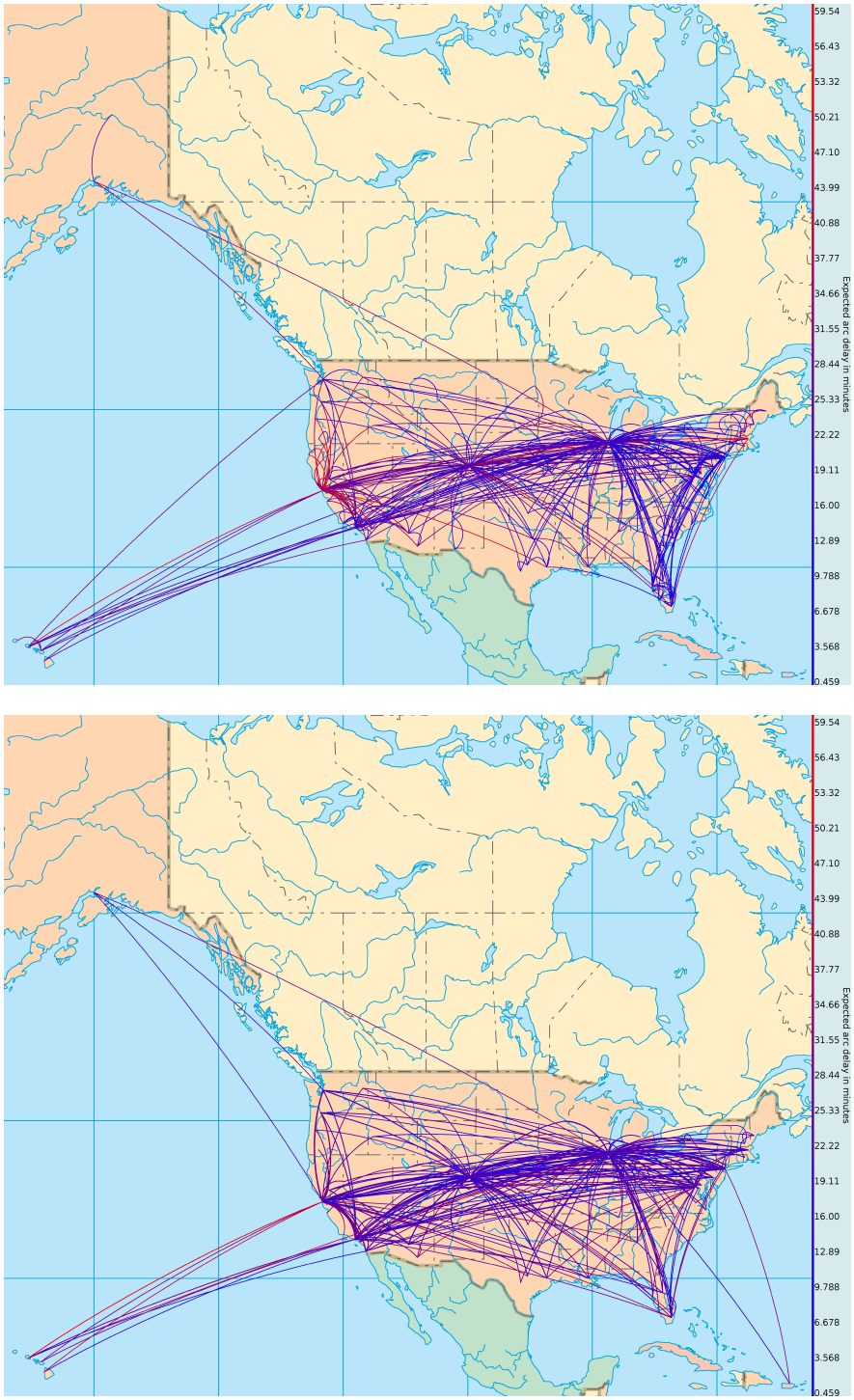


Figure 5.10: The United flight network; the top graph shows the 1987 network, while the bottom shows the 1997 network. Arc color indicates average observed delay.

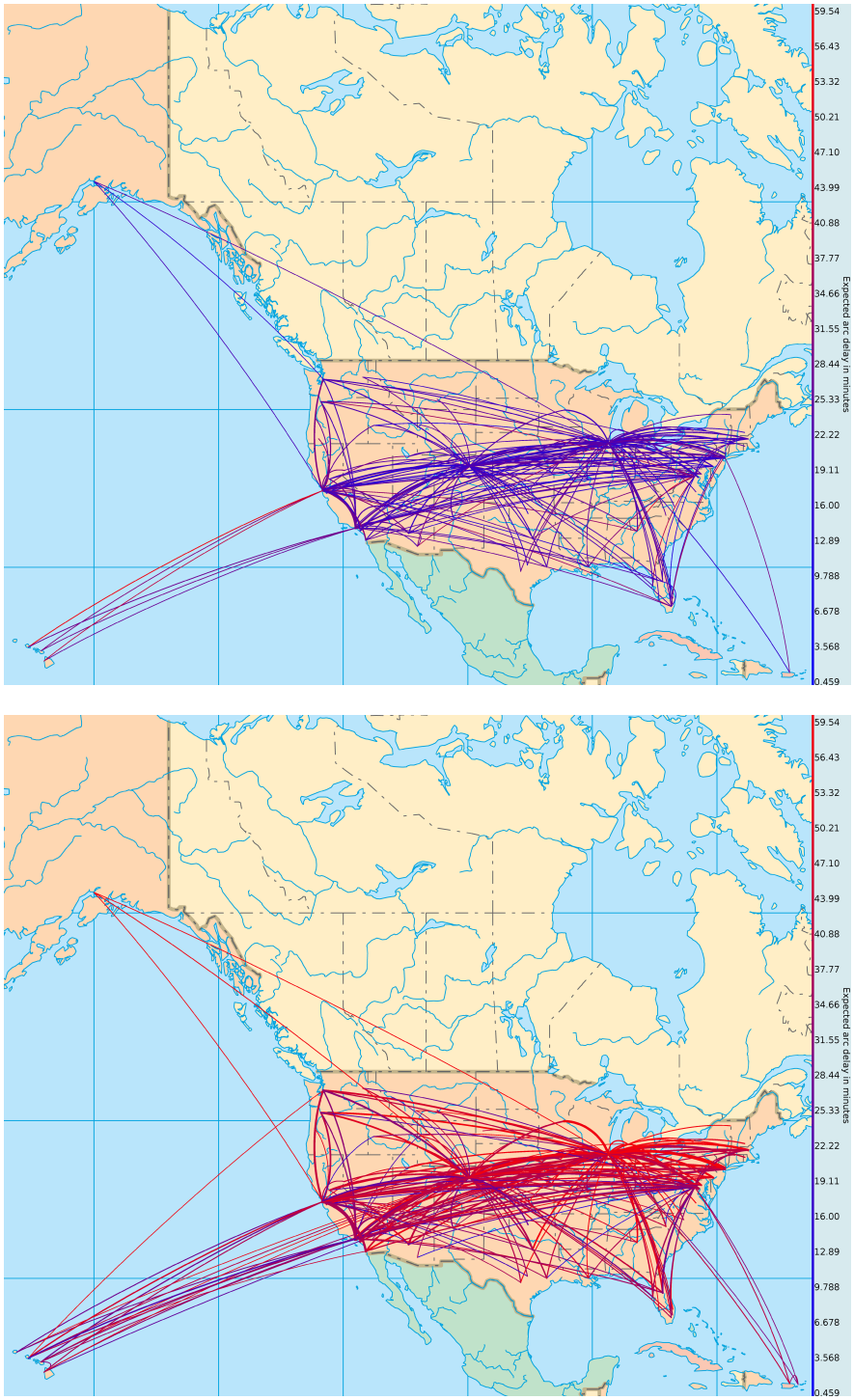


Figure 5.11: The United flight network; the top graph shows the 2002 network, while the bottom shows the 2008 network. Arc color indicates average observed delay.

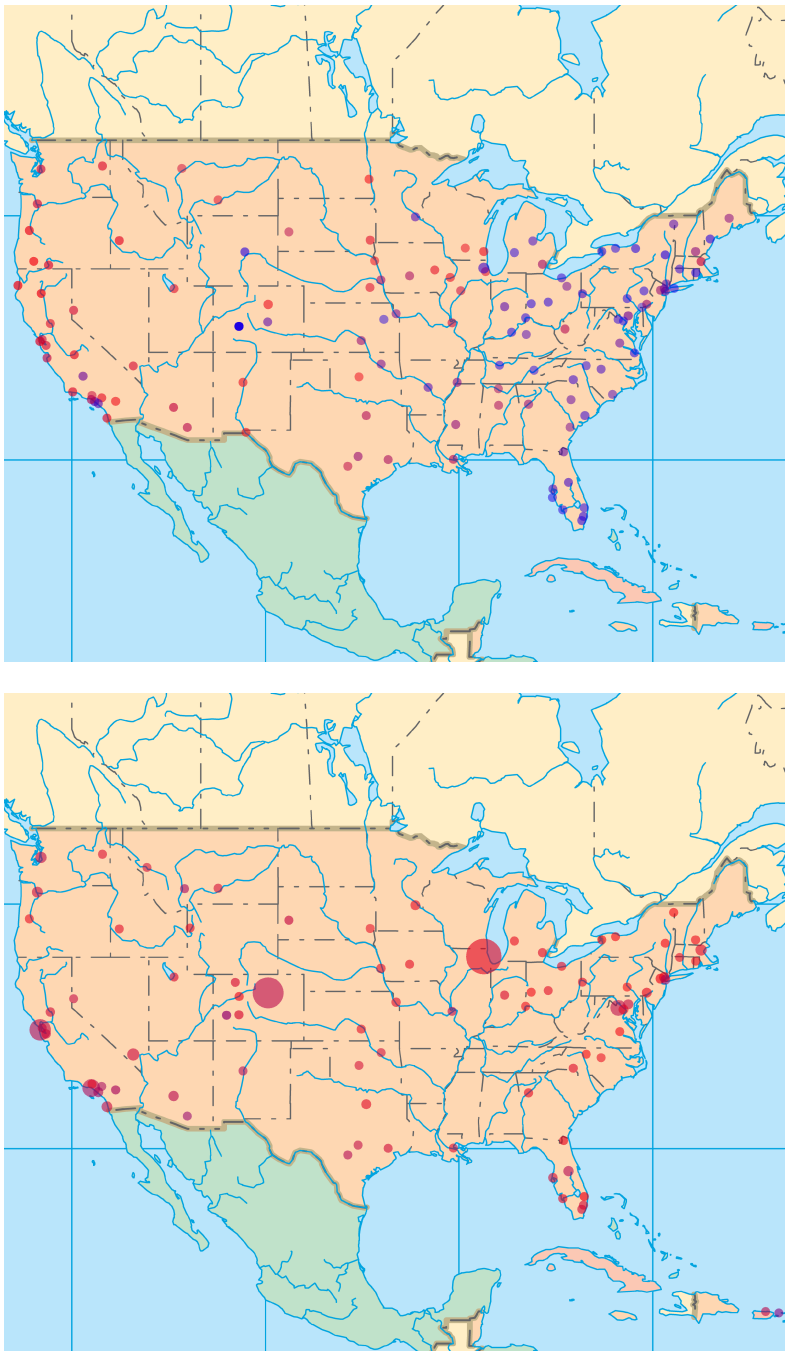


Figure 5.12: The United flight network; the top graph shows the 1987 network, while the bottom shows the 2008 network. Each circle represents an airport; the size of the circle represents the number of flights to and from the airport, the color of the circle represents the average delay time greater than 15 minutes for flights leaving the airport, and the opacity of the circles represents the probability of a flight leaving the airport being delayed more than 15 minutes.

between the smallest and largest airports is less pronounced in the Southwest network than in the United network, indicating that Southwest Airlines distributes its flights more evenly across its airports.

5.3 Total Distance Analysis

So far, we have only considered finding minimum delay flight paths; we now consider minimizing the total time of travel. To accomplish this, we use the mixed-weighting technique outlined in section 4.3.4; from equation (4.4), we choose $\theta_1 = \theta_2 = 1$ in order to weight air time and delay time equally.

Finally, we consider the problem of measuring the distance between airports with respect to total travel time. Although this could be accomplished by viewing many network maps showing the results of shortest paths calculations, this is cumbersome; instead, we create network maps such as the one in figure 5.14. Here, we have plotted a number of nodes, representing airports, around one central airport node – this central node is the origin airport. Each other node is placed a certain radial distance from the center corresponding to the total flight time required to travel from the origin to that node. The color of the node represents how many minutes of the total flight time are due to delay. The angle at which a node is located with respect to the horizontal axis carries no inherent meaning, although if one were to sweep out the area of the plot starting from some angle, one would encounter the nodes in the same order as if one were to sweep the globe from the origin airport.

5.4 Software

We created a visual support tool to allow our algorithms to be easily queried, as well as to graphically display the results. Aside from the histograms and box-and-whisker plots used to display summaries of the raw data set, all graphics in this paper were generated using our tool; the others were generated using the R statistical language ([7]). Over 12,100 lines

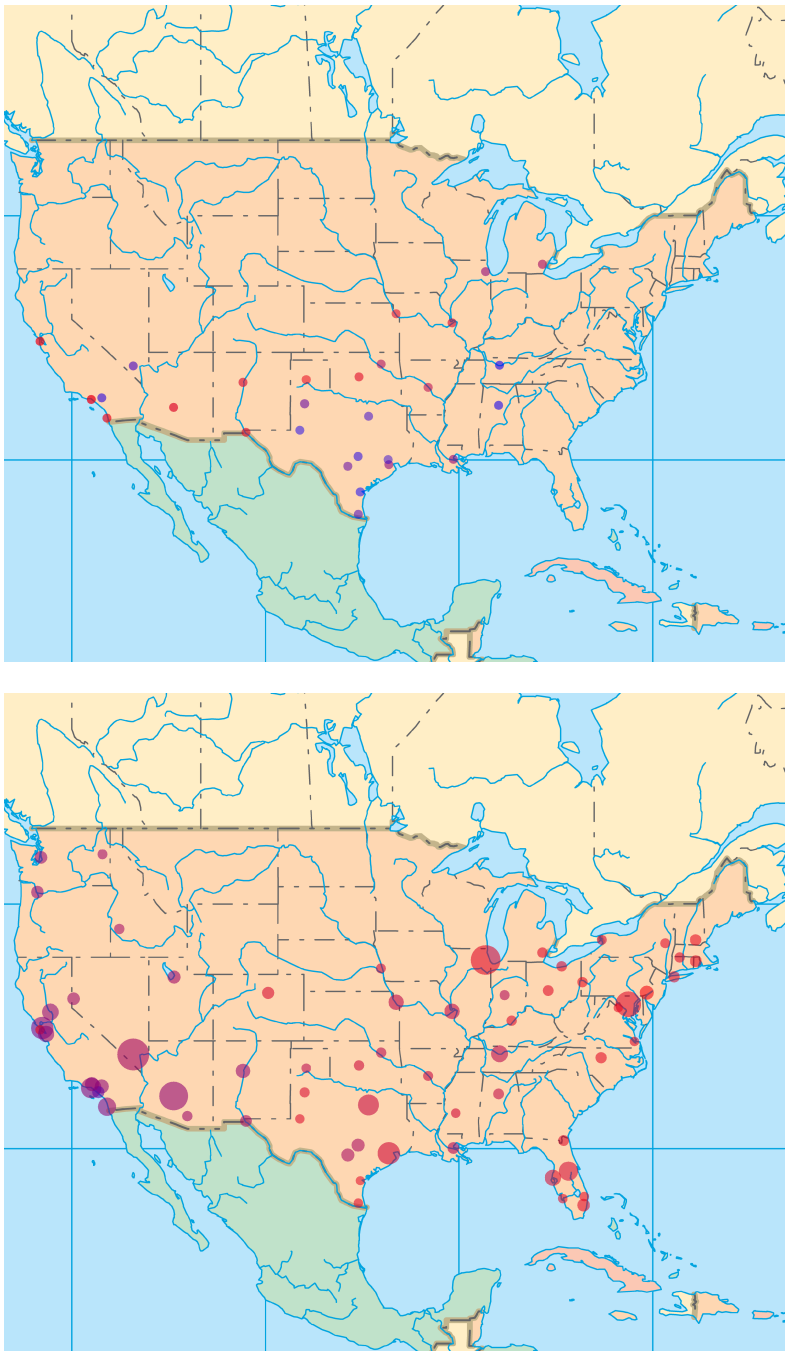


Figure 5.13: The Southwest flight network; the top graph shows the 1987 network, while the bottom shows the 2008 network. Each circle represents an airport; the size of the circle represents the number of flights to and from the airport, the color of the circle represents the average delay time greater than 15 minutes for flights leaving the airport, and the opacity of the circles represents the probability of a flight leaving the airport being delayed more than 15 minutes.

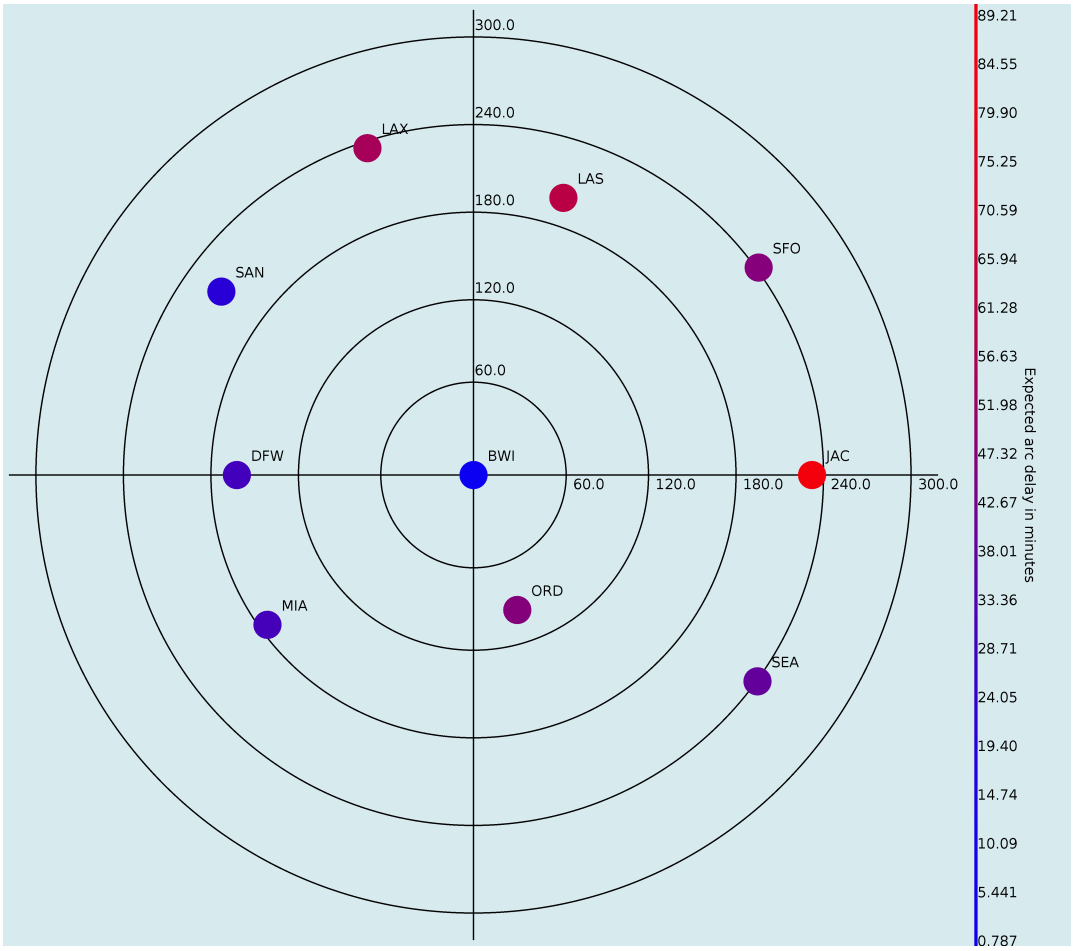


Figure 5.14: The total time required to travel to a number of different airports from Baltimore-Washington International airport. The distance of a node from the center represents the total time of required to travel from the origin to the node, while the color of the node represents how many minutes of that time are due to delay.

of Java, Python, C, C++, R, and Bash code were written to solve the shortest path problem, create the visualization tools, and perform auxiliary tasks; over 6,900 additional lines of code were generated by the previous code as well, bringing the total number of lines of code to over 19,000 for the project. All algorithms can be run through one of several front-end, fully documented command line interfaces. Included in these tools are several scripts designed to solve a large number of problems in a row and displaying the results graphically.

Much of the computation was performed on machines using two Intel(R) Core(TM)2 Duo CPUs clocked at 3GHz and 4Gb of ram. In addition, this work was performed in part using computational facilities at the College of William and Mary which were provided with the assistance of the National Science Foundation, the Virginia Port Authority, Sun Microsystems, and Virginia's Commonwealth Technology Research Fund.

5.4.1 Visualization

As has been demonstrated, the visualization software is capable of creating a number of different styles of network graphs. It is capable of modeling the solutions to shortest paths problems, the structure of flight networks, the volume of air traffic through airports, and spatio-temporal distances between airports. The graphics that our software creates are novel in that they present information in an additional spatial dimension by superimposing geographically-related information over the relevant map. The visualization software was written in Java, a widely supported language, allowing the software to be run without modification on many machine architectures.

5.4.2 User Interfacing

A secondary function of our software is its use as a clean user interface that allows for the use of all our algorithms. The software provides a graphical front end that allows a user completely unfamiliar with our work to utilize our results simply by filling in a series of

forms. To utilize our ability to find minimum delay routings, a user would simply select origin and destination airports from a menu, and fill in any qualifying parameters, such as the day or time of the flights of interest. The software then queries the necessary subroutines to solve the problem, and then interprets the results graphically in the form of one of the graphs previously discussed. This graph is displayed on screen, allowing the user to pan and zoom around the map to examine the results. In addition to handling user queries, the user interface is capable of loading, manipulating, and saving any graphics that our algorithms can generate.

Bibliography

[1] M. ARGÜELLO, J. BARD, AND G. YU, *A GRASP for aircraft routing in response to groundings and delays*, Journal of Combinatorial Optimization, 1 (1997), pp. 211–228.

[2] S. BRATU AND C. BARNHART, *Flight operations recovery: New approaches considering passenger recovery*, Journal of Scheduling, 9 (2006), pp. 279–298.

[3] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN, *Introduction to Algorithms, third edition*, MIT press, 2009.

[4] D. DELAHAYE AND A. ODONI, *Airspace congestion smoothing by stochastic optimization*, in Evolutionary Programming VI, Springer, 1997, pp. 163–176.

[5] S. LAN, J. CLARKE, AND C. BARNHART, *Planning for robust airline operations: Optimizing aircraft routings and flight departure times to minimize passenger disruptions*, Transportation Science, 40 (2006), pp. 15–28.

[6] A. NILIM, L. EL GHAOUI, AND V. DUONG, *Multi-Aircraft Routing and Traffic Flow Management under Uncertainty*, in 5th USA/Europe Air Traffic Management Research and Development Seminar, Budapest, Hungary, 2003, pp. 23–27.

[7] R. TEAM, *R: A language and environment for statistical computing*, Vienna, Austria: R Foundation for Statistical Computing, (2007).

[8] S. YAN, C. SHIEH, AND M. CHEN, *A simulation framework for evaluating airport gate assignments*, Transportation Research Part A: Policy and Practice, 36 (2002), pp. 885–898.

Appendix A

Summary of the Data

A.1 Key Statistics

	Flight Delay		
Carrier	Mean	Variance	Standard Deviation
American	37.05	13308.54	115.36
American Eagle	34.35	11507.38	107.27
Continental	34.86	12146.66	110.21
Delta	26.42	8357.61	91.42
Skywest	24.59	8142.47	90.24
Southwest	24.01	6042.07	77.73
United	37.48	13054.13	114.25

Figure A.1: The sample mean, variance, and standard deviation of the total flight delay for major carriers. The data is from 2005 to 2008.

American Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	11.16	1763.05	41.99
Departure	11.93	1463.63	38.26
Carrier	3.92	527.94	22.98
NAS	4.74	301.88	17.37
Security	0.02	1.01	1.00
Late Aircraft	5.29	446.63	21.13
Weather	0.93	81.45	9.02

Figure A.2: The sample mean, variance, and standard deviation of each component of flight delay for American Airlines. The data is from 2005 to 2008.

Continental Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	10.29	1631.85	40.40
Departure	10.96	1339.04	36.59
Carrier	2.87	347.55	18.64
NAS	6.59	523.11	22.87
Security	0.06	2.09	1.45
Late Aircraft	4.09	410.10	20.25
Weather	0.66	75.43	8.69

Figure A.3: The sample mean, variance, and standard deviation of each component of flight delay for Continental Airlines. The data is from 2005 to 2008.

Delta Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	7.54	1137.19	33.72
Departure	8.01	884.02	29.73
Security	3.11	322.76	17.97
Late Aircraft	4.23	235.21	15.34
NAS	0.01	1.22	1.10
Carrier	3.52	284.23	16.86
Weather	0.31	31.90	5.65

Figure A.4: The sample mean, variance, and standard deviation of each component of flight delay for Delta Airlines. The data is from 2005 to 2008.

American Eagle Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	10.70	1247.37	35.32
Departure	10.70	1247.37	35.32
Security	3.48	420.08	20.50
Late Aircraft	3.98	254.74	15.96
NAS	0.00	0.27	0.52
Carrier	5.80	415.23	20.38
Weather	0.82	94.27	9.71

Figure A.5: The sample mean, variance, and standard deviation of each component of flight delay for American Eagle Airlines. The data is from 2005 to 2008.

SkyWest Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	6.95	1077.68	32.83
Departure	7.61	912.01	30.20
Security	4.09	310.64	17.63
Late Aircraft	2.34	278.60	16.69
NAS	0.03	1.43	1.19
Carrier	3.57	319.82	17.88
Weather	0.56	76.13	8.73

Figure A.6: The sample mean, variance, and standard deviation of each component of flight delay for SkyWest Airlines. The data is from 2005 to 2008.

United Airlines

	Flight Delay		
Component	Mean	Variance	Standard Deviation
Arrival	10.51	1707.94	41.33
Departure	12.31	1431.47	37.83
Security	3.60	345.71	18.59
Late Aircraft	4.45	304.53	17.45
NAS	0.00	0.10	0.31
Carrier	6.60	640.11	25.30
Weather	0.41	44.27	6.65

Figure A.7: The sample mean, variance, and standard deviation of each component of flight delay for United Airlines. The data is from 2005 to 2008.

Southwest Airlines

Component	Flight Delay		
	Mean	Variance	Standard Deviation
Arrival	5.26	812.31	28.50
Departure	10.28	685.77	26.19
Security	1.81	91.38	9.56
Late Aircraft	1.34	68.03	8.25
NAS	0.03	1.02	1.01
Carrier	5.29	391.31	19.78
Weather	0.38	40.82	6.39

Figure A.8: The sample mean, variance, and standard deviation of each component of flight delay for Southwest Airlines. The data is from 2005 to 2008.

A.2 Frequency Distribution of Total Delay

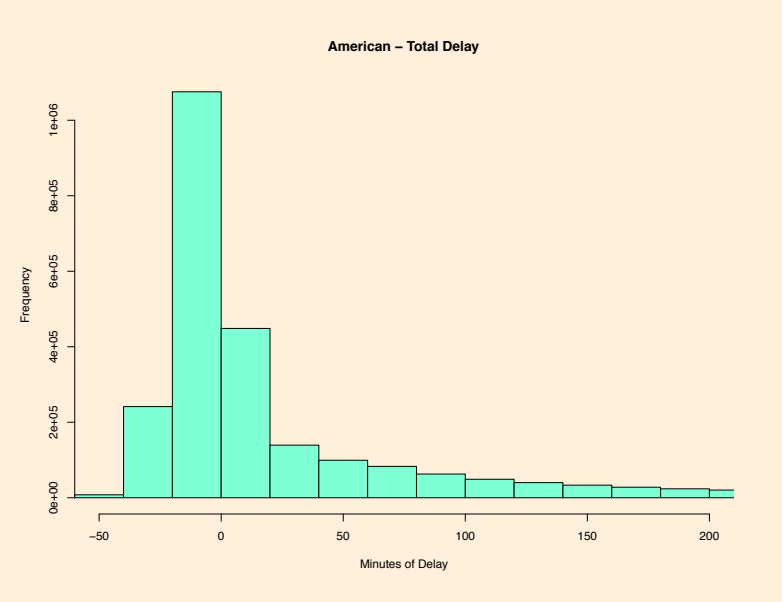


Figure A.9: A histogram plotting the frequency of total delay time. The data is for American Airlines from 2005–2008.

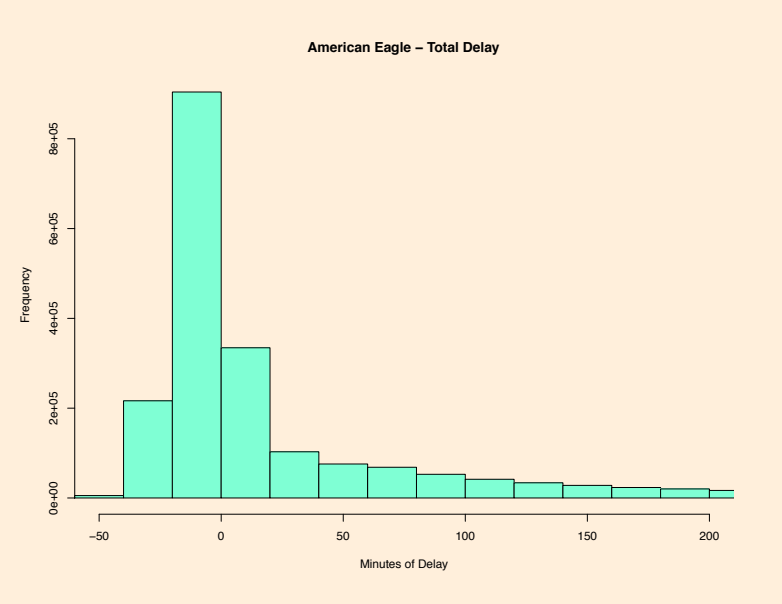


Figure A.10: A histogram plotting the frequency of total delay time. The data is for American Eagle Airlines from 2005–2008.

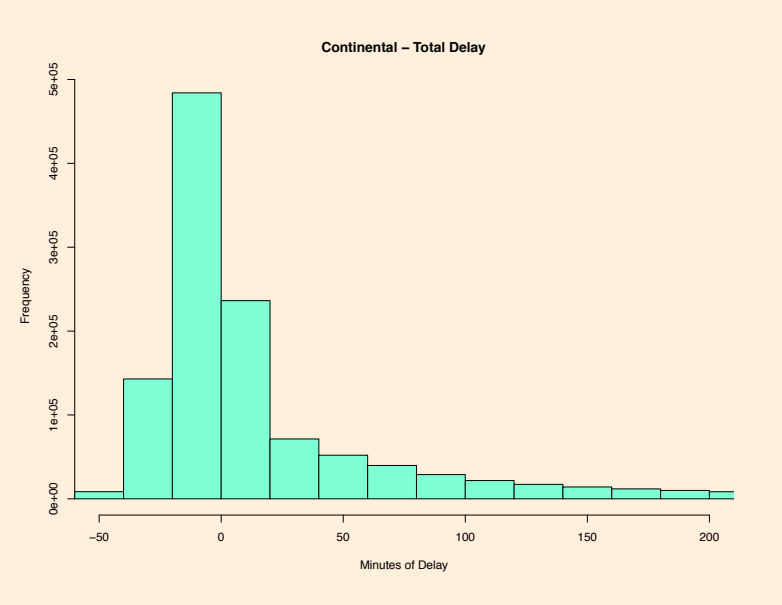


Figure A.11: A histogram plotting the frequency of total delay time. The data is for Continental Airlines from 2005–2008.

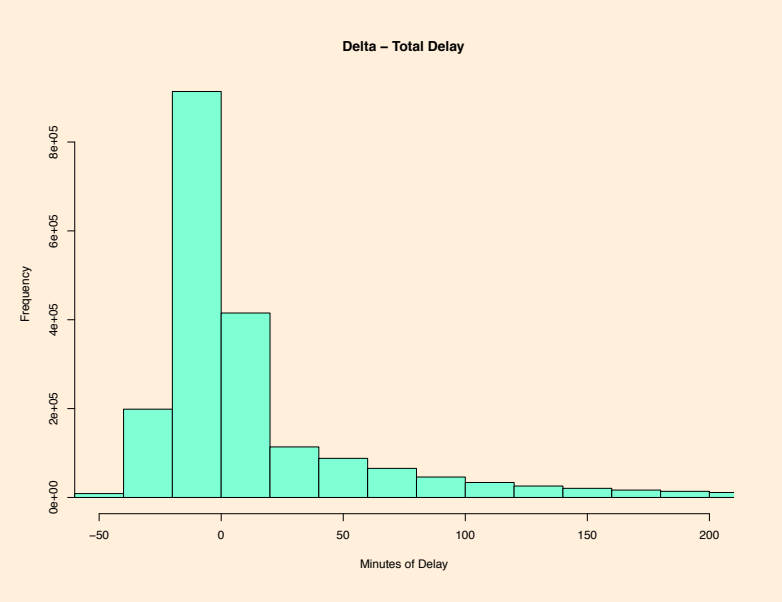


Figure A.12: A histogram plotting the frequency of total delay time. The data is for Delta Airlines from 2005–2008.

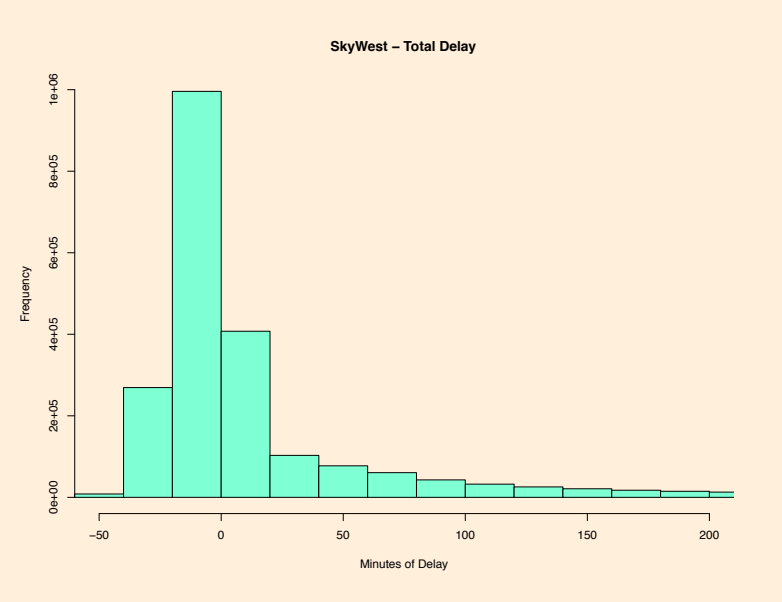


Figure A.13: A histogram plotting the frequency of total delay time. The data is for SkyWest Airlines from 2005–2008.

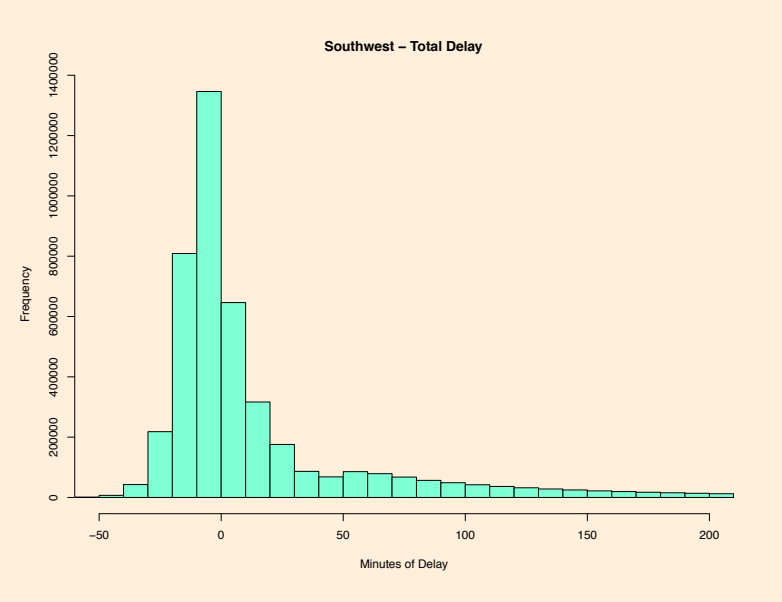


Figure A.14: A histogram plotting the frequency of total delay time. The data is for Southwest Airlines from 2005–2008.

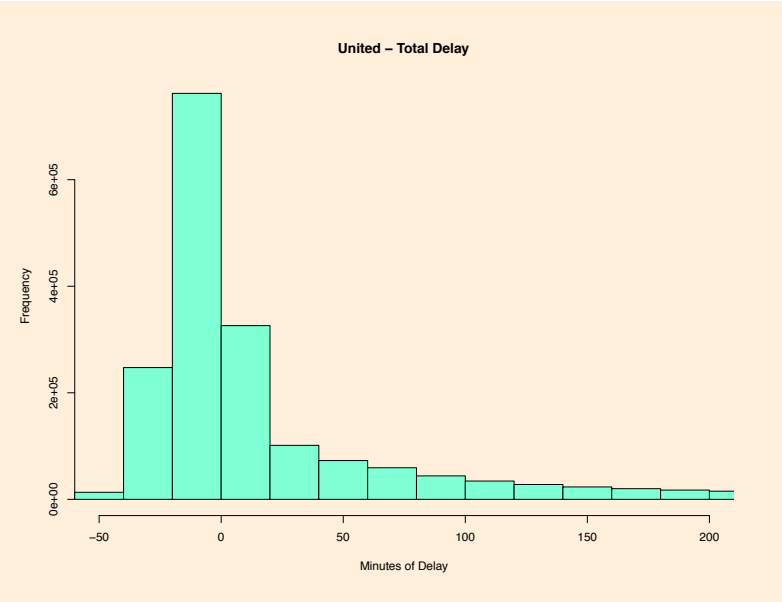


Figure A.15: A histogram plotting the frequency of total delay time. The data is for United Airlines from 2005–2008.

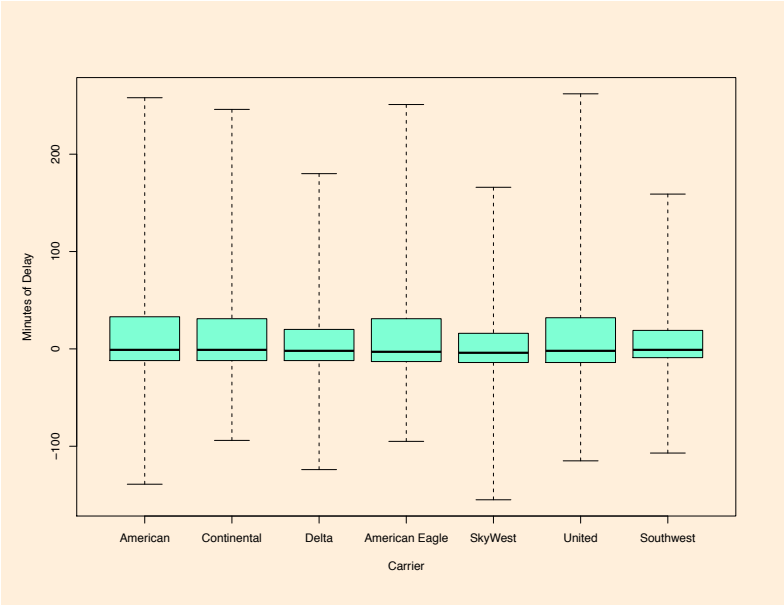


Figure A.16: A box-and-whisker plot of the total delay for major carriers. The data is from 2005–2008. Negative times indicate early arrivals.

A.3 Frequency Distribution of Delay Components

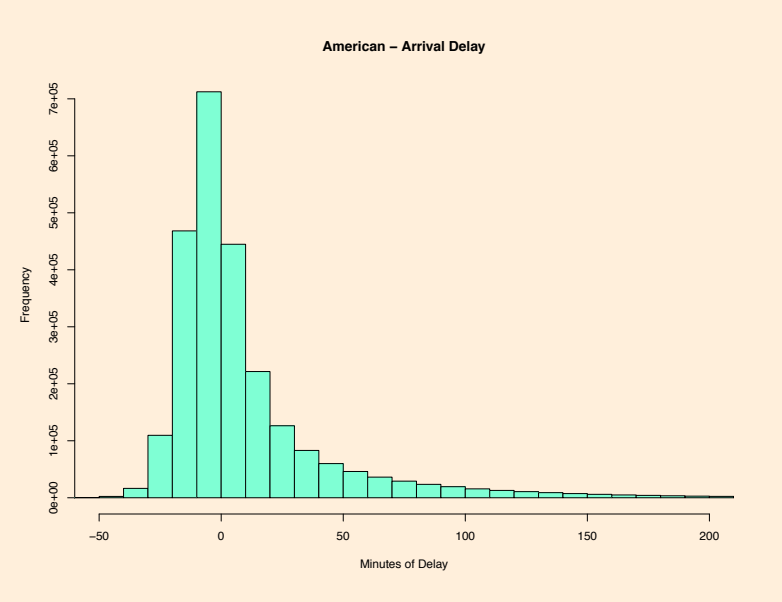


Figure A.17: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

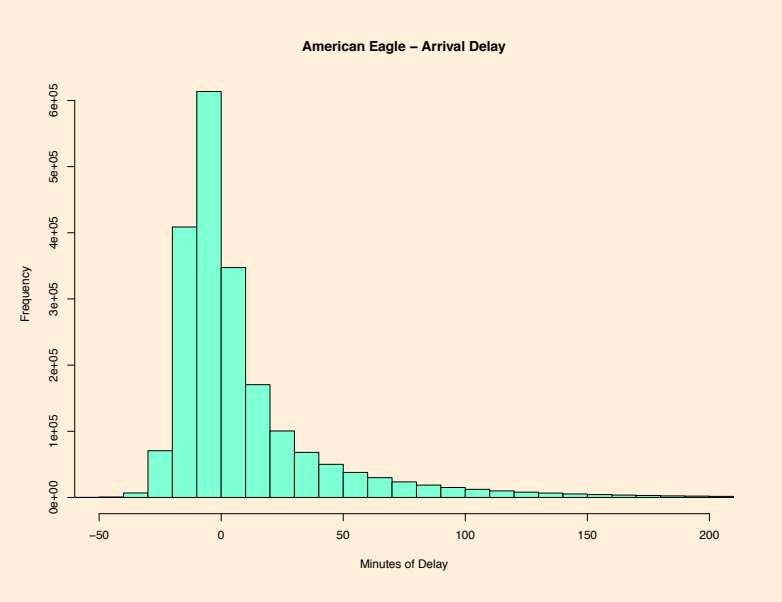


Figure A.18: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

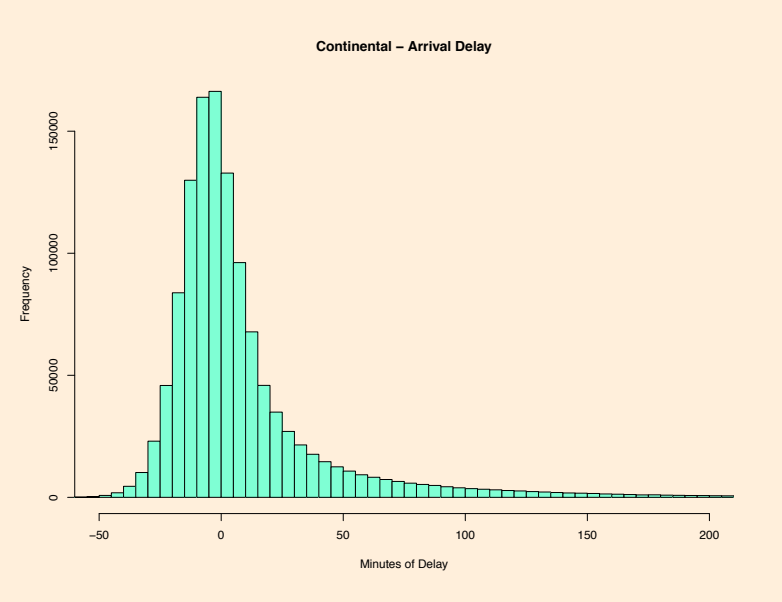


Figure A.19: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

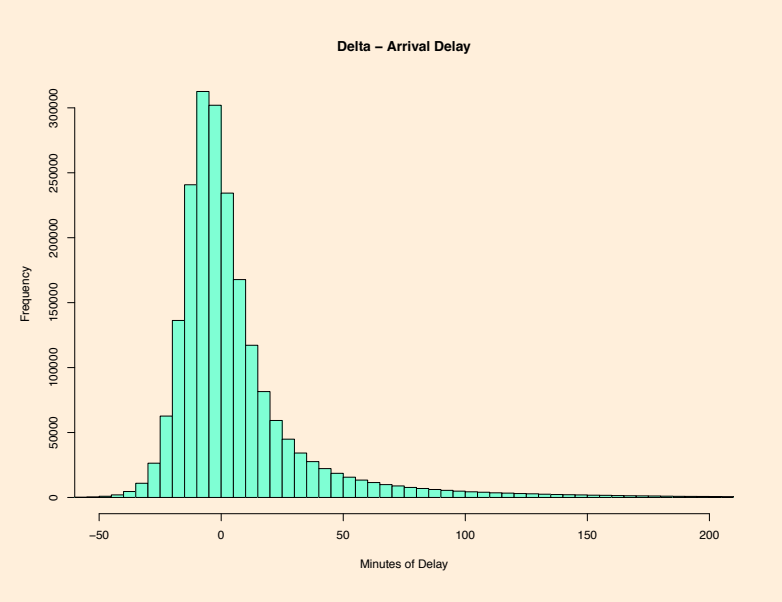


Figure A.20: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

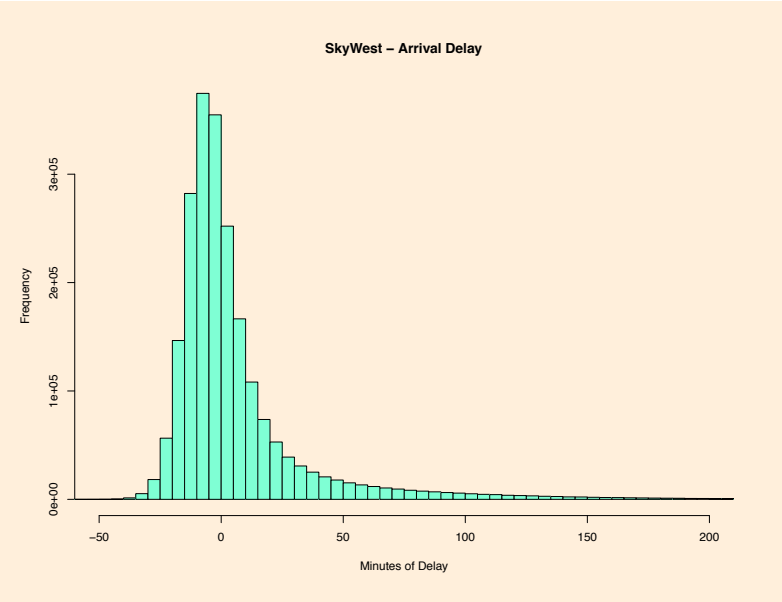


Figure A.21: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

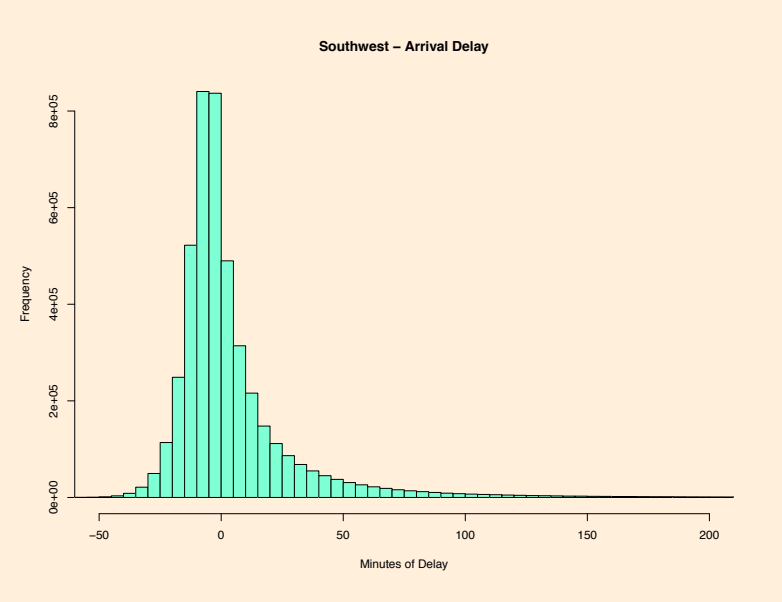


Figure A.22: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

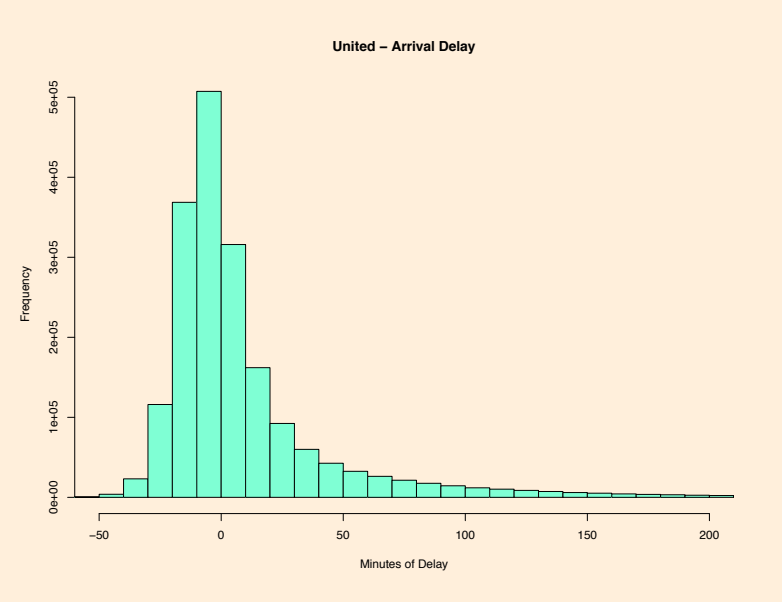


Figure A.23: A histogram plotting the frequency of arrival delay time. The data is for American Airlines from 2005–2008.

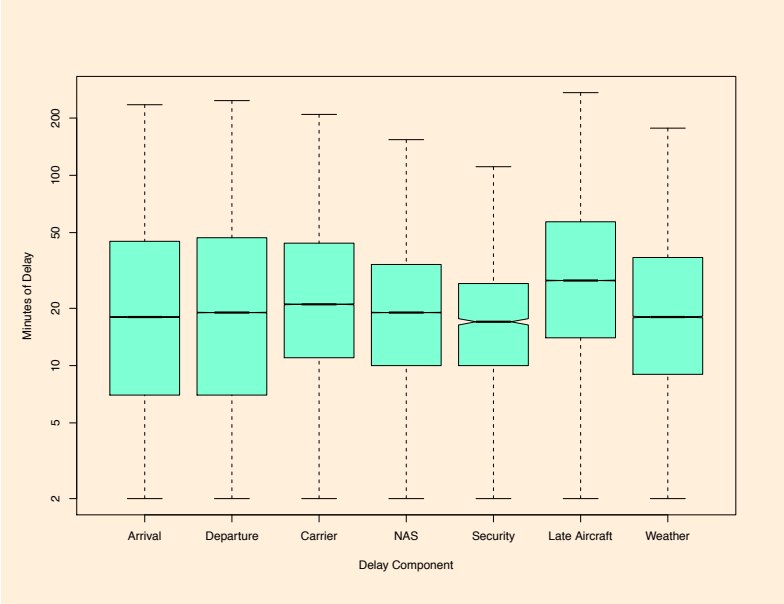


Figure A.24: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

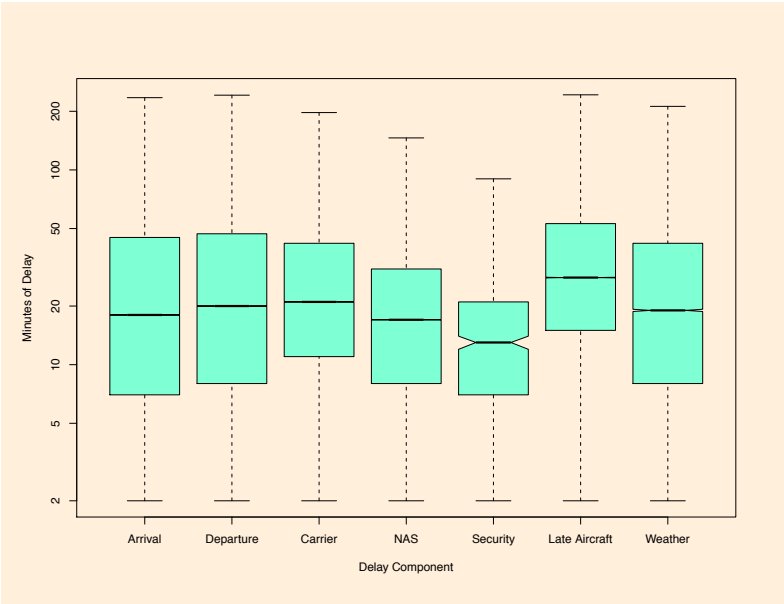


Figure A.25: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

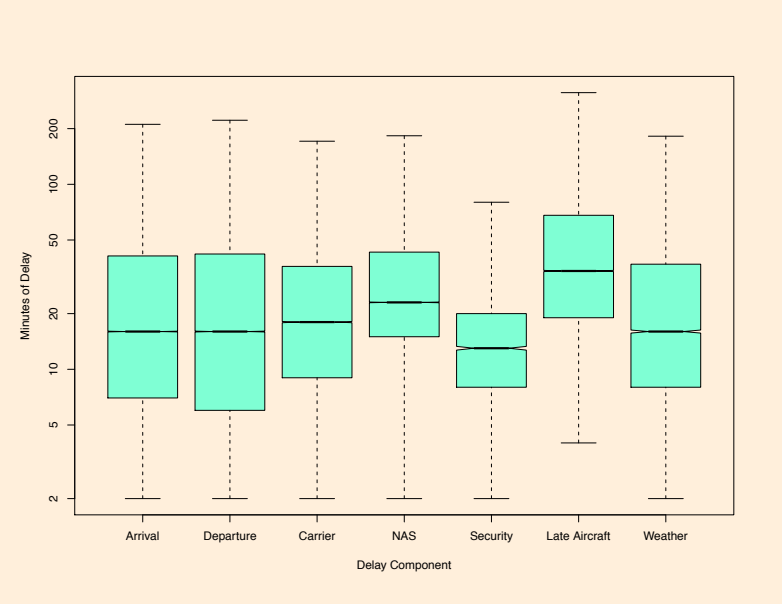


Figure A.26: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

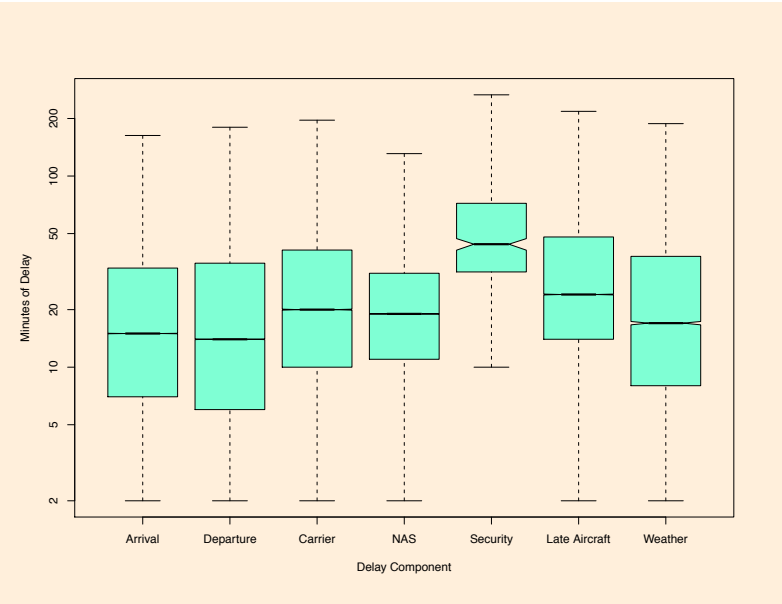


Figure A.27: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

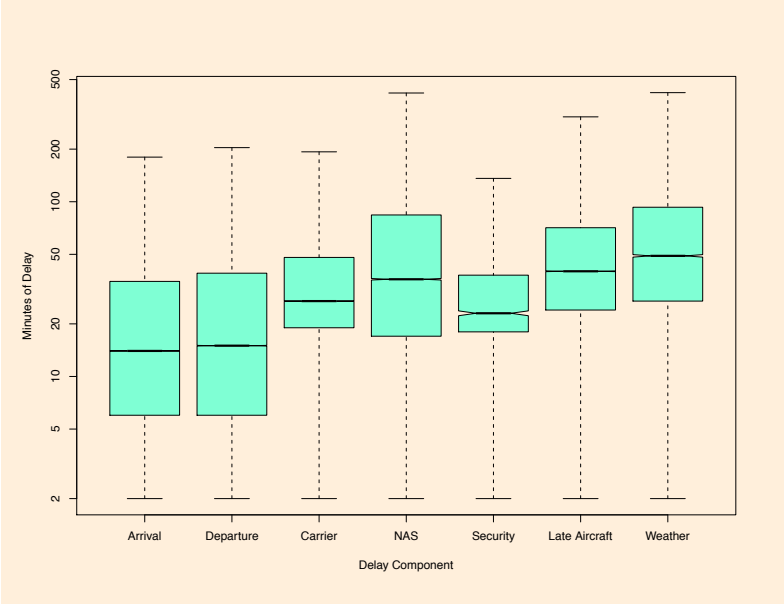


Figure A.28: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

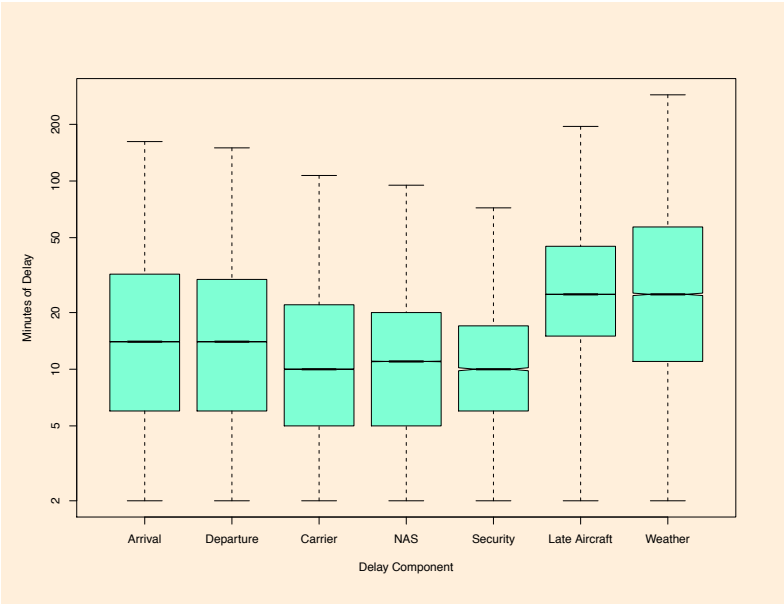


Figure A.29: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

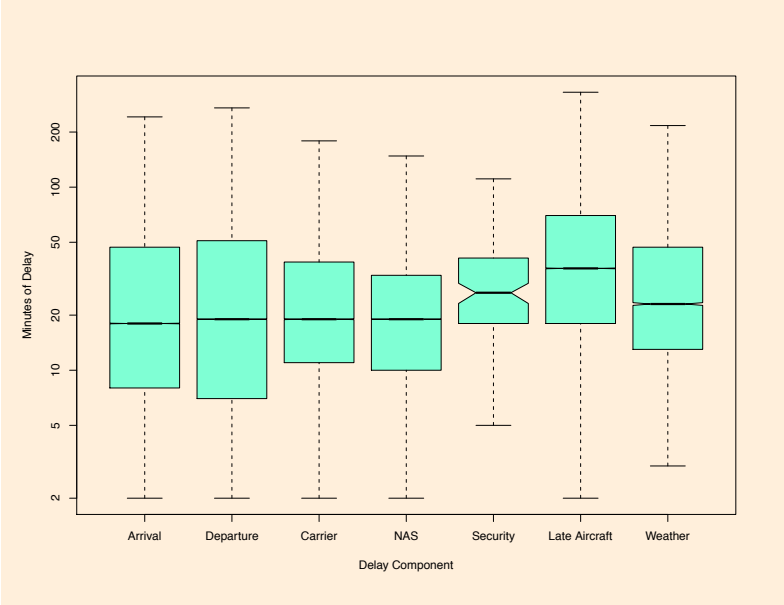


Figure A.30: Box-and-whisker plots showing the five number statistics for each component of delay for American Airlines. Nonpositive delays have been omitted, and the upper and lower bounds shown represent the largest and smallest values within 5 times the interquartile range, respectively. Note that the vertical axis has a logarithmic scale.

A.4 Correlations of Delay Variables

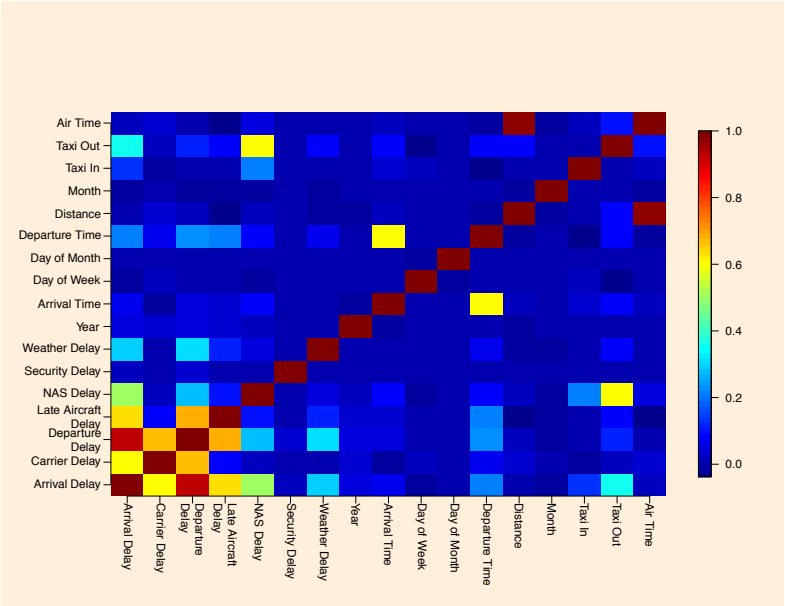


Figure A.31: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

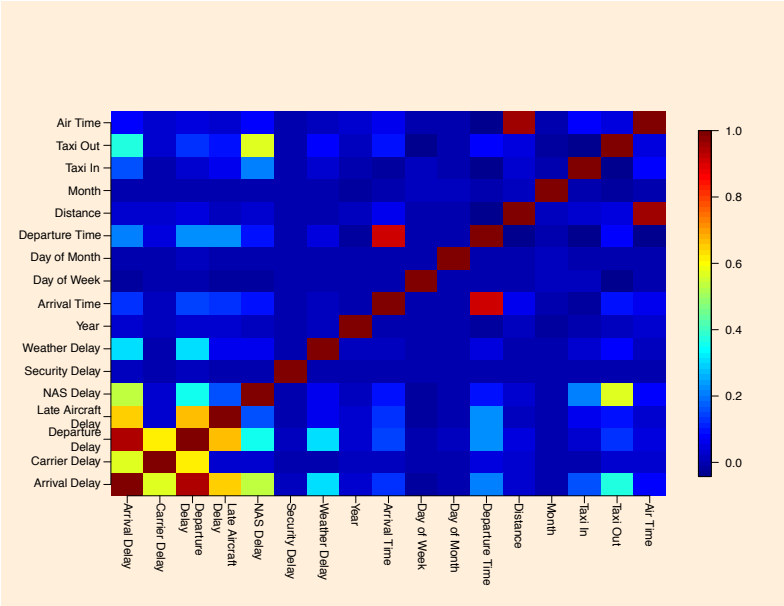


Figure A.32: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

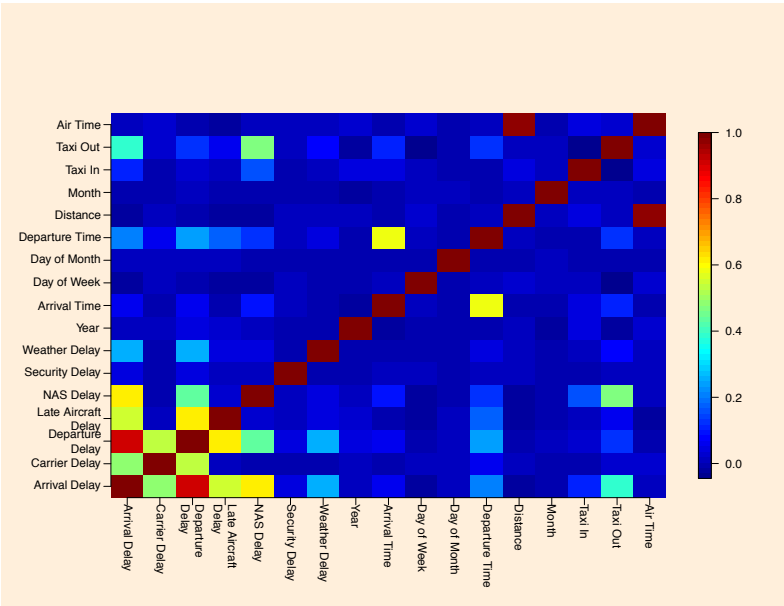


Figure A.33: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

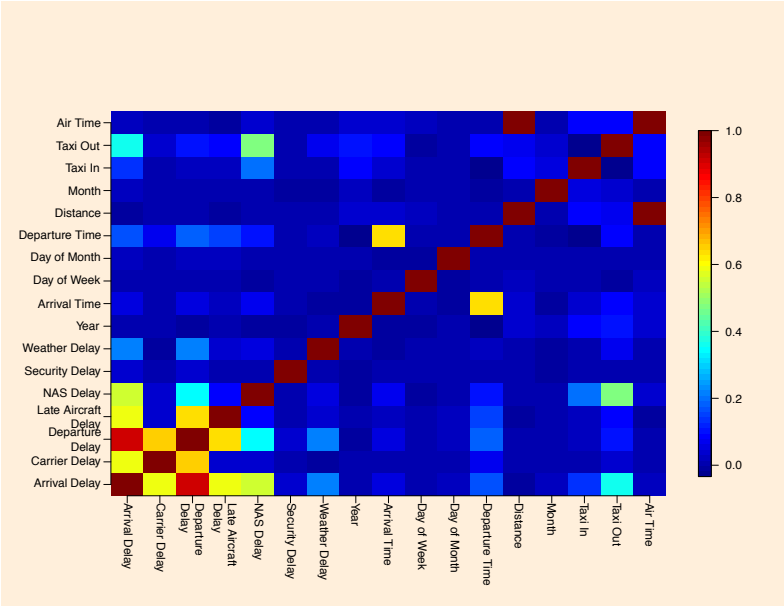


Figure A.34: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

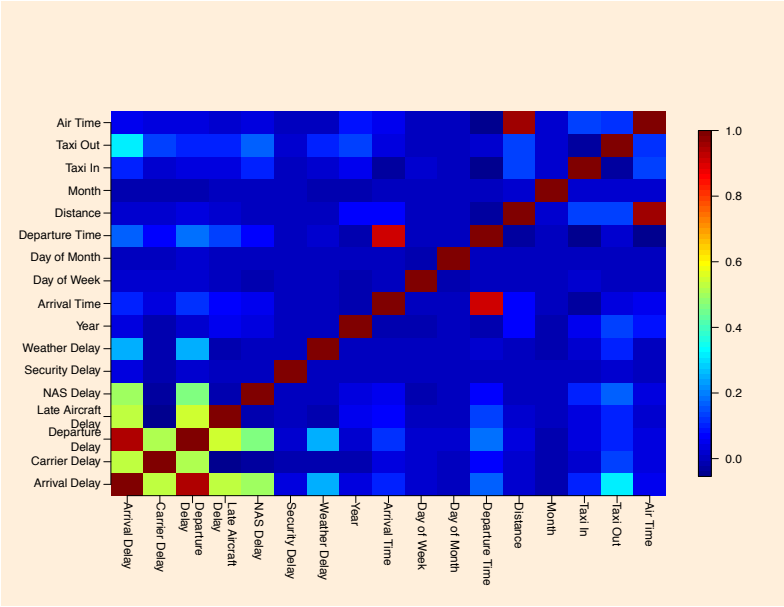


Figure A.35: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

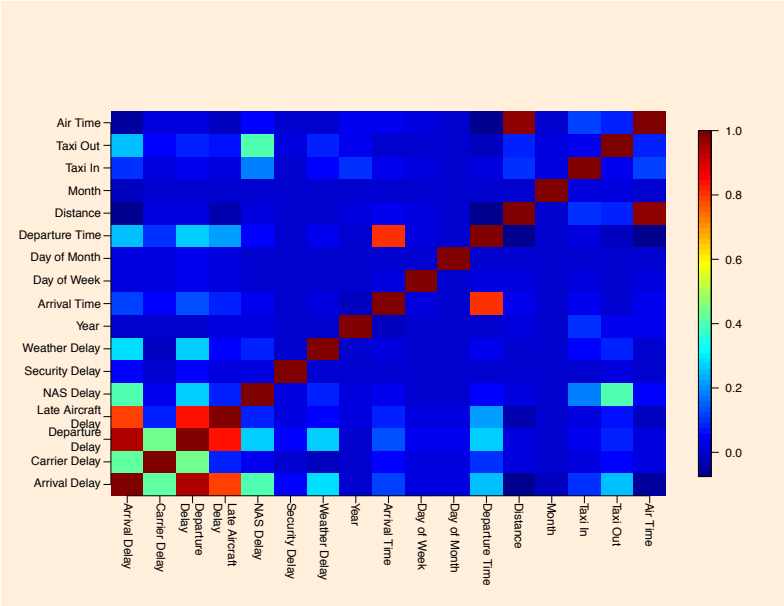


Figure A.36: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

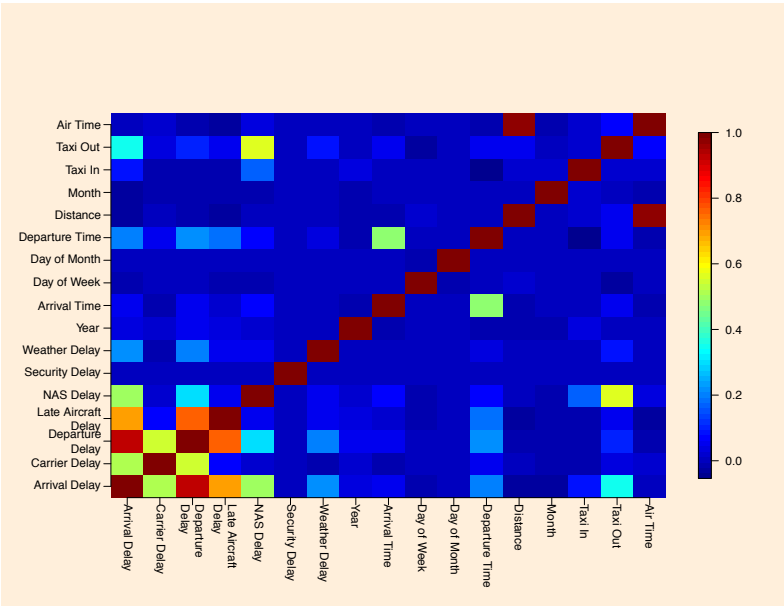


Figure A.37: The correlation matrix of the variables from the dataset; data is for American Airlines from 2005 to 2008.

Appendix B

Regression Coefficients

	American	Continental	Delta	American Eagle
Year	4.3945e-02	2.4828e-02	1.6338e-02	2.8802e-02
Arrival Delay	6.5614e-02	6.1126e-02	6.2883e-02	6.5742e-02
Arrival Time	1.3257e-04	2.0397e-04	2.4085e-04	3.1002e-04
Carrier Delay	-6.9956e-02	-5.8508e-02	-5.7357e-02	-6.5562e-02
Day of Week	4.4509e-03	-1.2112e-04	4.5640e-03	4.9384e-03
Day of Month	6.1634e-04	1.5126e-03	1.5503e-03	1.2205e-03
Departure Delay	2.1746e-02	1.7684e-02	1.8435e-02	2.0708e-02
Departure Time	4.2327e-04	2.9317e-04	1.7202e-04	1.2573e-04
Distance	8.6416e-05	8.4686e-05	8.2864e-05	2.1481e-04
Late Aircraft Delay	-6.0670e-02	-5.4799e-02	-5.1783e-02	-5.3242e-02
Month	-2.2380e-03	-7.8978e-04	5.7907e-03	3.0616e-03
NAS Delay	-5.5275e-02	-5.3341e-02	-4.6299e-02	-5.8998e-02
Security Delay	-4.9957e-02	-3.0636e-02	-5.3841e-02	-4.3234e-02
Taxi In	3.4368e-02	3.5729e-02	3.3986e-02	2.8277e-02
Taxi Out	2.9167e-02	3.2666e-02	3.0916e-02	3.4859e-02
Weather Delay	-6.6875e-02	-6.1243e-02	-6.2349e-02	-6.8949e-02

Figure B.1: The $\boldsymbol{\beta}$ estimators from the multiple linear regression for American, Continental, Delta, and American Eagle airlines.

	SkyWest	United	Southwest
Year	1.6909e-02	4.6315e-02	-1.0532e-02
Arrival Delay	5.7504e-02	5.9307e-02	5.4081e-02
Arrival Time	2.1373e-04	1.4961e-04	2.0542e-04
Carrier Delay	-4.1667e-02	-5.6513e-02	-3.4020e-02
Day of Week	1.1115e-02	8.3993e-03	2.3586e-03
Day of Month	1.5286e-03	1.1547e-03	-6.8676e-04
Departure Delay	2.1392e-02	2.2170e-02	2.6683e-02
Departure Time	5.6327e-05	3.0444e-04	2.4447e-04
Distance	7.9751e-05	5.8751e-05	1.1590e-04
Late Aircraft Delay	-4.4282e-02	-5.5537e-02	-4.4795e-02
Month	1.2631e-03	2.0882e-03	-5.1246e-03
NAS Delay	-5.2019e-02	-5.0990e-02	-3.8289e-02
Security Delay	-2.6697e-02	-3.2444e-02	-1.6274e-02
Taxi In	3.5249e-02	3.2483e-02	3.2383e-02
Taxi Out	3.5106e-02	3.1797e-02	3.2971e-02
Weather Delay	-5.4707e-02	-6.4862e-02	-5.8201e-02

Figure B.2: The $\boldsymbol{\beta}$ estimators from the multiple linear regression for SkyWest, United, and Southwest airlines.