

# SANDIA REPORT

SAND97-1806 • UC-722

Unlimited Release

Printed July 1997

## Creating Route Characterization Files Using the NHPN Database

Mathias J. Sagartz

Prepared by  
Sandia National Laboratories  
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia  
Corporation, a Lockheed Martin Company, for the United States  
Department of Energy under Contract DE-AC04-94AL85000.

Approved for public release; distribution is unlimited.

**Sandia National Laboratories**

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

**NOTICE:** This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from  
Office of Scientific and Technical Information  
P.O. Box 62  
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from  
National Technical Information Service  
U.S. Department of Commerce  
5285 port Royal Rd  
Springfield, VA 22161

NTIS price codes  
Printed copy: A03  
Microfiche copy: A01

## **Creating Route Characterization Files Using the NHPN Database**

Mathias J. Sagartz  
Experimental Structural Dynamics Department  
Sandia National Laboratories  
P. O. Box 5800  
Albuquerque, New Mexico 87185-0557

### **Abstract**

The ADROIT code was written to perform route specific analyses of the risks involved in highway transportation of nuclear materials. This report describes a companion code, Route-Tool, that has been written to provide ADROIT with the route characterization files it requires. These files define a route as a series of points. Associated with each point are five pieces of data: two coordinates (latitude and longitude), an operating environment parameter, a meteorological station, and a route mile marker. The NHPN (National Highway Planning Network) database that Route-Tool uses for its primary data source is described. Features of the NHPN that are important for route characterization are discussed and a detailed explanation of the steps involved in creating route files is presented. Tools and techniques that the code provides for editing and, as necessary, supplementing the NHPN data are discussed. The report is intended to provide a user familiar with the ADROIT code all the information needed to create new route files.

## **Acknowledgments**

Financial support for this work was provided by the Transportation Safeguards Division, DOE Albuquerque Operations Office. David Clauss provided general guidance and a large part of the graphical user interface code used in Route-Tool. His help is gratefully acknowledged.

# Contents

<b>1 Introduction</b>	<b>7</b>
<b>2 NHPN Database</b>	<b>7</b>
<b>3 Generating a Route File, an Overview</b>	<b>8</b>
<b>4 General Requirements</b>	<b>10</b>
<b>5 Generating a Route File, the Details</b>	<b>11</b>
5.1 Preliminary Preparations . . . . .	11
5.2 Entering Highway s..... . . . .	12
5.3 Editing the Highway Data.. . . . .	13
5.4 Putting Links in Geographic Order . . . . .	16
5.5 Adding aHighway to the Route.. . . . .	17
5.6 Combining all the Highway Data for a Route . . . . .	17
5.7 Collecting Data and Creating the Route File . . . . .	18
<b>6 Comments and Excuses</b>	<b>22</b>
<b>7 References</b>	<b>24</b>
<b>A Matlabizing the NHPN Database</b>	<b>25</b>
<b>B Creating an End Stub</b>	<b>26</b>
<b>C Assigning Meteorological Station Codes</b>	<b>28</b>
<b>8 Distribution</b>	<b>30</b>

## List of Figures

1	Graphics Window Display for 1-40 in Arizona . . . . .	10
2	Loop in140 in Arizona . . . . .	14
3	Region Delete in Colorado . . . . .	15
4	Plot of a Complete Route . . . . .	19
5	Meteorological Station Map . . . . .	20
6	Code Architecture . . . . .	23

## List of Tables

1	Meteorological Station Codes . . . . .	28
---	--	----

# 1 Introduction

The ADROIT code was written to perform route specific risk assessments for highway transportation of nuclear materials. To accomplish this task ADROIT requires that the route(s) involved in the analysis be adequately characterized. Specifically, ADROIT uses route characterization files that provide it with the following five pieces of information at each point along the route:

- Location - the latitude (1) and longitude (2) of the point.
- Operating environment (3) - a code that classifies the location as urban or rural and the highway access as either controlled or not controlled.
- Meteorological station (4) - a code number of the meteorological station whose historic weather database is used to describe the statistical distribution of weather conditions at the point.
- Route mile marker (5) - the distance from the start of the route to the point.

There are no fixed rules about the number of points needed to characterize a route, but standard practice is to use all the information a data source can provide, i.e., the more the better.

A description of the method used to generate route characterization files when ADROIT was first developed is presented in Reference [1]. Data sources used were the Bureau of the 1990 Census TIGER/Line files and the 1990 Census PL94-171 population and housing data. The process involved was not highly automated and was very labor intensive. To simplify and automate the generation of route files, a new procedure using a different data source has been developed. For consistency with ADROIT and to minimize the user software requirements, all coding was done as MATLAB scripts. MATLAB is a “High-Performance Numeric Computational and Visualization Software” package available from The Math Works, Inc.

## 2 NHPN Database

Data for the first three of the five pieces of information needed at every point on a route are available in the U.S. Department of Transportation’s National Highway Planning Network (NHPN) database. The remaining two pieces can easily be generated once the first three have been obtained. Important advantages of the NHPN over the census data are that the required information is contained in a single database, it is relatively small and highway specific, and it has higher geographic resolution. Also, the NHPN contains more specific highway information than the census data so additional data are available should they be required by a future version of ADROIT. All data in the NHPN are stored as simple ASCII files which can be easily and quickly obtained by downloading

(anonymous ftp) from the Department of Transportation's internet site, ftp.dot.gov. The database files are stored in the *pub/fhwa/gis/nhpnv202* directory.

Data in the NHPN files are organized on a state-by-state basis. For each state, the database contains three files that together characterize all of the major roads and highways in that state. These files are known as the link, node and geo files. The NHPN breaks up all highways into segments called links. Links define pieces of a highway having a particular set of attributes such as type of access control, divided or undivided, highway name or designation, number of lanes, whether a link lies in a large urban, small urban or rural area, etc. Data for each link are contained in one, 121-character record in the link file. State link files typically contain between two and four thousand records, but Texas has over eight thousand. For referencing purposes, each link record contains a unique ID number. The only information required for route characterization that is lacking in the link files is geographic position. That is provided by the state's other two files, the node and geo files.

End points of links are called nodes and like links, nodes have ID numbers by which they can be referenced. Each link record contains the ID numbers of its nodes or end points. An NHPN node file consists of one, 83-character record for each node in a state. Record contents consist of information such as node ID, geographic location (latitude and longitude), and why a node exists at that location; e.g., county boundary, intersection of two highways, highway attribute changes, etc. A key characteristic of the database for use in route characterization is that the intersection of two highways occurs at a node. A single node is always located where two NHPN highways intersect and that node will be referenced by links belonging to both intersecting highways.

Geo files provide detailed geographical location information for all of a state's highways. Node data provide only the geographical location at the end points of links, but the length of a link can vary from a small fraction of a mile to 40 or more miles. The geo file includes a group of records for each link that define the detailed geometry of that link. For a single link, the geo file can contain from 2 to 500 latitude/longitude coordinate pairs, depending on the length of the link and its geometric complexity.

### **3 Generating a Route File, an Overview**

The task of creating a route file is essentially one of assembling a geographically ordered list of link ID's for all of the NHPN links that make up the route. Geographically ordered means that a vehicle traveling the route would roll over the links in exactly the order they appear on the list. Once this list is defined, it is a simple matter to open the appropriate link and geo files and to access and parse the required records to obtain latitude, longitude, and operating environment for every point on the route. With the latitude and longitude known, the correct meteorological station can be assigned and the route mile marker calculated. A common complication to this process is the necessity of using end stubs on some routes. End stubs are short segments of minor highways, not

included in the NHPN database, that are used by the route to get from either the origin or the destination to a major road or highway included in the NHPN. A description of how to create end stubs is given in Appendix B.

Unfortunately, the operations involved in generating the route file are not clean enough to be accomplished without active user intervention. Route-Tool is an interactive code designed to facilitate the required user interactions. The user interface is a hybrid, requiring both text input and usage of a GUI (graphical user interface). Active participation by the user is needed primarily because of the different ways the NHPN and the user view a highway. To the NHPN, a highway includes alternate routes, business bypasses, loops, and spurs while the user thinks of it as piece of an actual route, which is a specific, unambiguous, path. The principle task for the user is to edit the NHPN highway data to eliminate all extraneous links that cause ambiguities and to insure that the list of highway links includes all of the links that are part of the route. Except for the highways at the ends of the route, the user is not expected to specify where the route joins a particular highway and where it leaves that highway. The code does that automatically. However, the code has no way of determining where the route begins and ends unless the user indicates those locations by deleting all of the links that extend beyond the ends of the route for both the first and last highways.

To create a route file, the user starts MATLAB and enters the program name, `Route_Tool`. The script prints out some introductory information and then prompts the user to enter the post office two-letter abbreviation for the state and the name of the first highway on the route in a standard NHPN format. The program loads in the state link file and uses the highway name in a string matching scan of the link records to find and save those that belong to that particular highway. The selected highway link records are then parsed to generate a list of the record ID's for the nodes that belong to the links. The NHPN node file for the state is loaded and the list of node ID's is used to select the appropriate records. These records are parsed to find the coordinates for all of the nodes on the highway and to create a geographic plot of the highway links for the user to inspect. Figure 1 shows a sample plot of highway links for Interstate 40 in Arizona.

The code applies a consistency test to the highway data using the node ID information. Passing this test ensures that the highway links and nodes define a unique, unambiguous, geographic path. For highway data that initially fail the test, the script contains tools that can be used to edit and modify the highway information so that it will pass. The code does not allow the user to continue until the data pass the consistency test, and once this has been accomplished, he is asked to confirm that the highway should be added to the route. At that point highway data are saved as a matlab "mat" file.

The highway entry process is repeated for each highway belonging to the route. Because the NHPN is organized on a state-by-state basis, a highway that crosses a state boundary must be entered as a separate highway in each state. For example, to travel from Albuquerque, NM, to Amarillo, TX, would involve only Interstate 40, i.e., one highway but in two states. To create a corresponding route file requires that 1-40 be treated as two separate highways, 1-40 in New Mexico and 1-40 in Texas. Order is important when

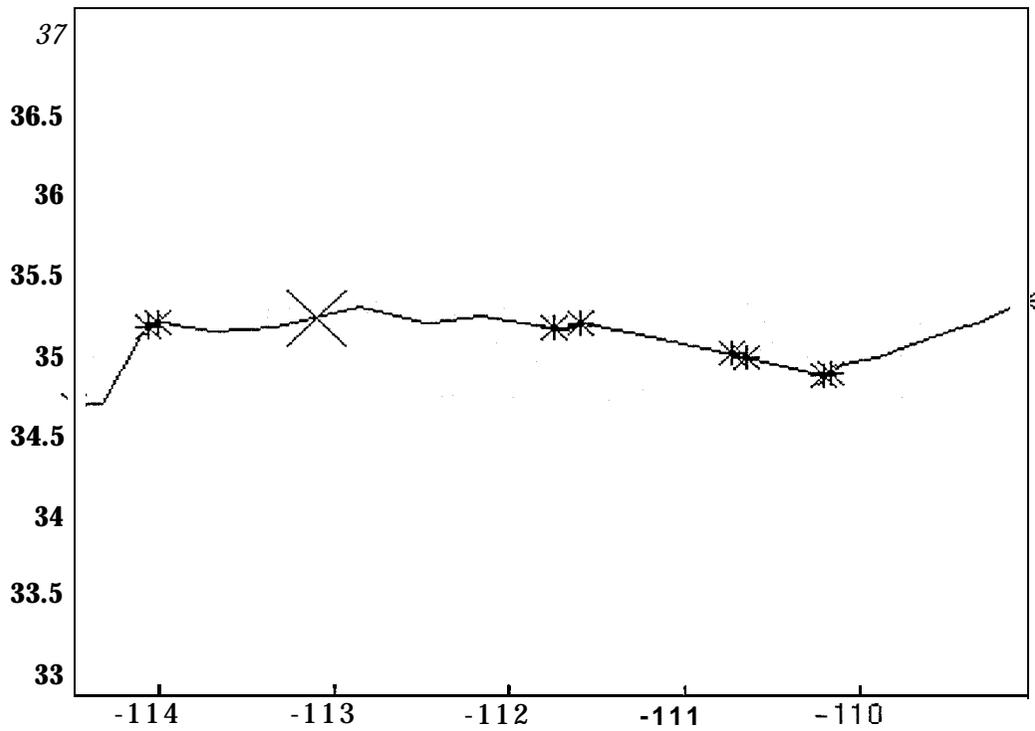


Figure 1: Graphics Window Display for 1-40 in Arizona

entering highways; they must be entered just as they are encountered along the route. After each highway has been processed, the user is asked whether or not he wishes to add another highway. A '(no" answer ends highway input.

Once all of the highways have been entered, the script automatically pieces together the route using link and node information from each highway. When finished, a geographically ordered list of links for the complete route is saved to a file and the user is shown a plot of the entire route. At that point the user is asked to confirm that he wishes to continue. If he chooses to continue, he is prompted to name the route file and then to provide a text description of the route to save with the file. Once the processing is finished, the user is given an opportunity to append end stubs and then the final route file, complete and ready for use by ADROIT, is saved.

## 4 General Requirements

Route\_Tool is coded in the MATLAB scripting language so MATLAB must be installed on the user's computer system. It was developed using MATLAB version 4.2c

on a SUN workstation. The NHPN database is also required, but it must be converted into a “matlabized” form by reprocessing with the scripts described in Appendix A. A copy of the NHPN documentation file should be obtained along with the data files. Reading the documentation and having it available while running the code can help the user understand the process and the meaning of information output to the screen. MATLAB expertise is not required, but some familiarity with MATLAB, graphical user interfaces, and file requesters is expected. Appropriate maps or a road atlas are very convenient things to have handy when creating route files.

## **5 Generating a Route File, the Details**

The process described above seems simple enough, but the old cliché, “The devil is in the details” certainly applies when it comes time to actually create a route characterization file. The NHPN database was developed as a general data source and adapting it for route characterization use requires considerable editing, modifying, and processing of its data. Also, the NHPN database is not perfect. Through experience we have discovered some inconsistencies and unexpected gaps in data for some highways. The routes analyzed by ADROIT tend to avoid the minor secondary roads that may not be included in the NHPN, but end stubs are a common complication that must be handled with data obtained from other sources. Stub files are essentially short route files that include the part of a route from an end location to a major highway. They must contain the same data in the same format as a complete route file and the stub end point must match up with a node on the major highway. By definition, stub roads are not included in the NHPN, so stub files must be created with an entirely different process from the one used to create the major part of the route file. Using a stub file on a route simply involves appending a very short route file to the front or back end of the file created from NHPN data.

### **5.1 Preliminary Preparations**

First and foremost, the user needs to have a complete knowledge of the route he wants to characterize. If there is some uncertainty about whether or not a particular highway is included in the NHPN, `Route_Tool` can be used as a preprocessing, scoping tool to search the database and determine what highways are available under what names. The user simply enters the state and highway name at the prompts and `Route-Tool` will report whether it found any data. The process can be repeated as often as desired simply by always asking to process another highway. If any end stubs are needed, these should be generated prior to processing the route. Appendix B explains the details of creating stub files. If the route does not need end stubs, the latitude and longitude coordinates of the end points must be known.

## 5.2 Entering Highways

When the Route-Tool script is started, it prints out a welcoming message and provides some instructions on how to enter the state and highway names, notifies the user that the order that highways are input is important, provides information on how to handle highways that cross state boundaries, and notes that travel from west-to-east or south-to-north is considered positive. The user is then prompted to enter a state abbreviation, which the script converts to an FIPS<sup>1</sup> code that it uses to load in the state link file. Another prompt requests the user to enter the highway name. If the highway is the first highway, for a route, the user is also asked whether or not the route uses a beginning stub. If the answer is yes, a file requester appears and the user is asked to select the appropriate stub file.

An NHPN highway name consists of a single letter followed by the highway route number, or it may simply be a street name. The more common single letter codes are: I for interstate, U for US highway, S for state highway, and C for county route<sup>2</sup>. As examples, the NHPN name for US Highway 287 is U287 and for State Road 219 it is S219. Name input is always padded on the end with blanks to create a string at least five characters long which is then used to search through the link data for substrings that match the highway name. The reason for padding to five characters is to prevent finding matches from US Highway 288 data when searching for US Highway 28 data. All link records providing a match are saved in a string array and a message is displayed to indicate how many matches, i.e., links, were found. Experience shows that this process is a reasonably reliable technique for locating all the links for a highway. Unfortunately, the NHPN often also considers alternate routes and spurs, which are usually not part of the route of interest, as part of the highway. These are also found in the string matching process and contribute extraneous links to the array.

After the link array is defined, the individual link records are parsed to generate a list of node ID's for all of the starting and ending nodes. If the highway links form a simple, unambiguous path, each node ID in the node list should appear twice, once as the starting node of a link and once as the ending node of an adjacent link. The only exceptions are the nodes at the end points of the highway which should each appear only once. If more than two nodes appear only once or if any nodes appear more than twice, the user is notified that the highway data failed a consistency test and need to be corrected before they can be used.

Normally, graphical methods are used to edit and correct the data, but there is an exception. Route-Tool plots links as straight lines connecting their end points so if two links have the same end points they overlay each other and can't be distinguished on the plot. The code checks for this situation and if it occurs, prints out the link records for both links and asks the user to select the one to be kept. However, this does not occur often and in the more common cases, a prompt is printed asking the user whether he wants

---

<sup>1</sup>FIPS codes are numerical codes used to identify states and urban areas.

<sup>2</sup>For a complete list, consult the NHPN documentation which can be **downloaded** from the same site as the database itself.

to continue or to quit, in which case the script terminates. Continuing involves using the tools built into the script to graphically edit the data until they pass the consistency test. Route-Tool will not let the user continue beyond this point until the highway data pass the test. Should the data pass without editing, the user is not asked if he wants to continue, but the code still takes him into the editing process at least once to give him a chance to inspect a plot of the links. It also provides an opportunity to delete links in the first and last highways that lie beyond the route end points.

### 5.3 Editing the Highway Data

**Once** the user has indicated a desire to continue, the script loads the state node file and uses the node ID list to select records that correspond to the nodes belonging to the links of the highway being processed. These records are then parsed to find the latitude and longitude coordinates for the nodes. A graphics window is opened, and a plot showing the links as straight lines connecting their end points is displayed. Also on the plot, nodes that appear only once are shown as green stars and those appearing more than twice are marked with red stars. If the highway is the first one on a route and an end stub has been selected, the script draws a large X to mark the node where the stub ends. The top border of the graphics window includes a Zoom check box, a Whoops! button, and two menus labeled *Overlays* and *Options*. These items provide tools for the user in editing highway data. Figure 1 shows a plot that has the end of a stub marked at approximately  $-113.1^\circ$  longitude.

The most common form of editing involves getting rid of bifurcations, or internal loops in the highway, by deleting links. Figure 2 shows a plot of such a loop taken from the 1-40 data in Arizona. To delete a branch, the user selects its links by positioning the cursor over each of them and clicking the left mouse button. Selecting an individual link often becomes easier if the region around the link is magnified. Clicking on the Zoom checkbox activates the MATLAB zoom function<sup>3</sup>, which enables the user to magnify a region of the plot. Once the magnification level is satisfactory, zoom should be clicked off while a link is being selected. Once selected, the link changes color and part of the link record is displayed in the text window. An unlimited number of links can be selected with this method. If the user changes his mind and decides he wants to keep one of the selected links, a second click on the link will unselect it and its color will be reset. When a link is selected, it is advisable to inspect the data displayed in the text window to provide some assurance that the branch of the bifurcation that is being deleted is really what the user wants to eliminate. If there is any doubt about which branch to eliminate, selecting links on both branches lets the user compare the link data displayed in the text window and come to an informed decision. Then clicking a second time on the links to be kept will unselect them.

Sometimes rather than selecting and deleting one link at a time, it is convenient to delete a group of links at once. For example, if only a fraction of the highway is used by

---

<sup>3</sup>Consult MATLAB documentation or online help for details of how the zoom function works.

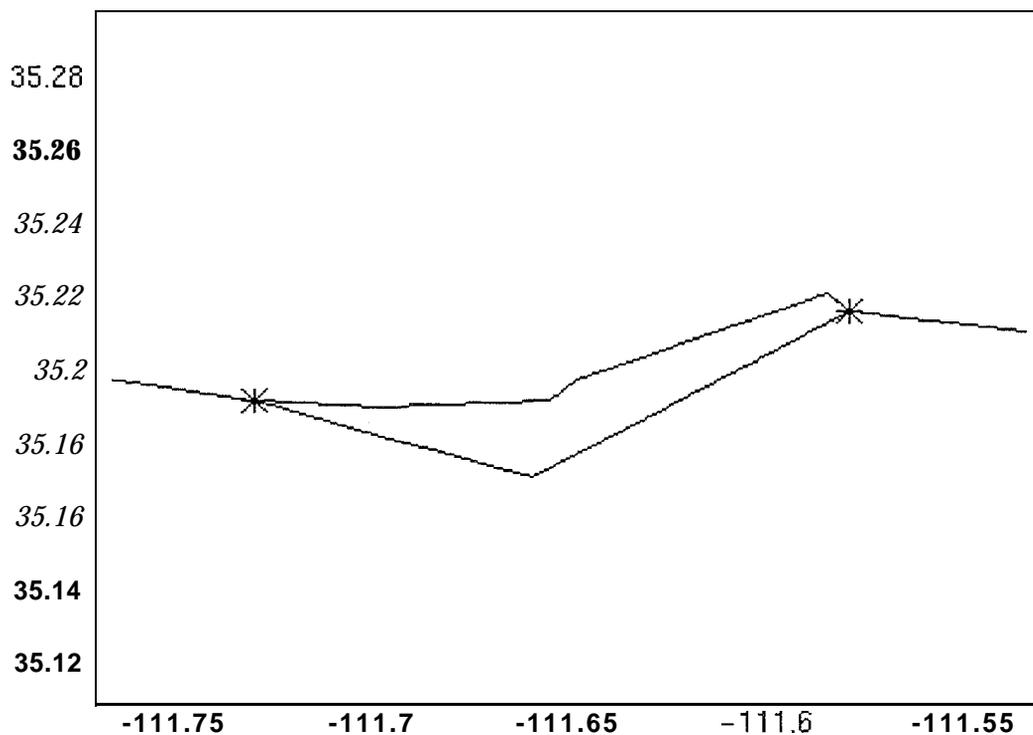


Figure 2: Loop in 140 in Arizona

the route, the user should not have to correct consistency problems in data that belong to sections of the highway that are not part of the route. In this case the user should open the *Options* menu and select *Region Delete*. A prompt will instruct him to select an origin by clicking on a point, i.e., move the cursor into the graphics window and click the left mouse button at a point that is convenient for defining a region. A large cross appears at that point and the user is instructed to click with the left mouse button in a quadrant of the cross. Doing so will select all of the links that lie completely in that quadrant for deletion and they will change color. Note that the quadrants defined for a region extend out indefinitely and are not limited by the size of the cross. If instead of clicking with his left mouse button, the user clicks with his right mouse button, the region delete will be canceled and no links are selected.

With just a simple straight line plot of links, visualizing which parts of the highway are actually part of the route can be difficult. This can make using *Region Delete* much less convenient. Well, not to worry, oh intrepid route characterize! Help is available from the *Overlay* menu. The *Political Boundaries* item allows the user to overlay state, county, urban, and metropolitan area boundaries on the plot, providing perspective and a sense of scale. Another source of help is the *Plot Previous Highway* menu item, which draws small circles at the nodes of the last highway added to the route. Of course, this option is

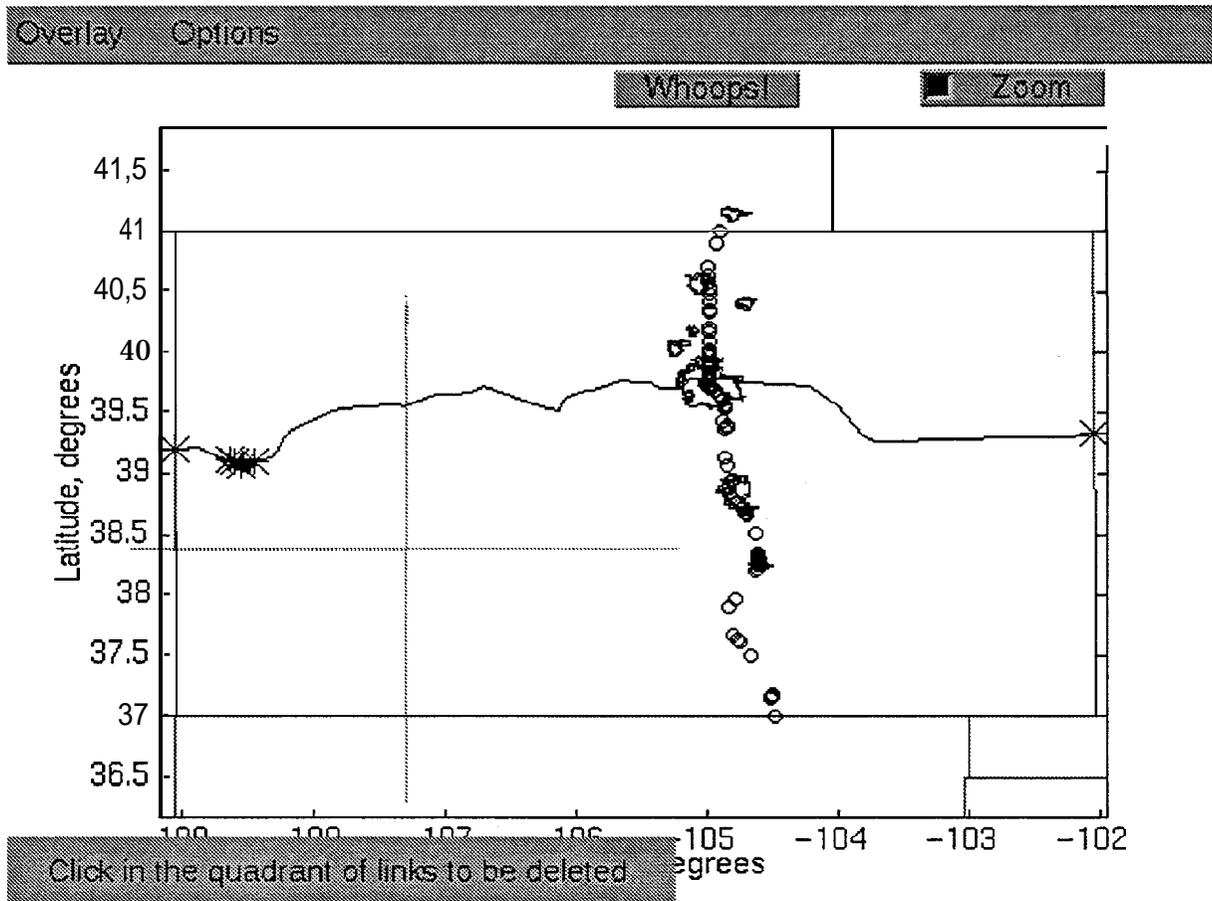


Figure 3: Region Delete in Colorado

of little help when dealing with the first highway on a route that does not use a starting stub. In that case the *Mark a Location* menu item can provide help. That option brings up editable text boxes to enter the longitude and latitude of a point that will be marked with a circle on the plot. By inputting the starting point of the route (when editing the first highway) or the ending point (when editing the last highway) the user can then use a Region Delete to get rid of parts of a highway that extend past the start or endpoint of the route. Figure 3 shows an example of a region being deleted from I-70 in Colorado with state and urban area boundaries plotted, and I-25 shown as the previous highway.

Except for the first and last highways, keeping links that lie beyond the part of any highway used by the route is perfectly acceptable. However, for the first highway the user must edit out all links that extend beyond the starting point of the highway, and on the last highway the links beyond the ending point of the highway must also be deleted. With interior highways, the route following capabilities of Route\_Tool will automatically decide which links are not used and discard them.

Occasionally, highway data will contain gaps that do not exist in reality. The first step in resolving a gap problem is to check available maps for the possibility that the highway simply changes its name over a region. In that case the list of highway routes

can be amended and the gap can be filled by entering the highway under its corrected name. If the gap cannot be crossed using links found under another name, the last resort is the *Create a Link* item in the *Options* menu. This will prompt the user to click on the two points to be joined, whereupon, a straight line segment is drawn between them. Then clicking the left mouse button will accept the new segment and clicking with any other mouse button will reject it.

## 5.4 Putting Links in Geographic Order

When the user is finished editing, he must move the cursor back into the text window and press return. This signals the code to use the edits to delete from or add to the highway link and node arrays and to repeat the consistency test on the modified data. If the data still do not pass, the user is again asked if he wants to continue. The editing process can be repeated as many times as necessary until the highway data pass the consistency test. There is a Whoops! button at the top of the plot window that allows the user to discard current edits and start over. Pressing the *Whoops!* button undoes all of the editing done since the last time return was pressed, but edits done prior to the last return press cannot be undone.

After the consistency test is satisfied, the script calls the `firstnode.m` function, which chooses one of the two highway end points as the beginning point for travel along the highway in a positive sense. This endpoint is marked with a red circle and the user is asked if travel starting at the marked endpoint and ending at the other endpoint traverses the highway in the same direction as the route. The user's answer establishes the direction of travel for the route on that particular highway.

Once the starting node has been selected, the script locates the link containing this node and stores it as the first link for the highway. It then finds the node ID number of the node at the opposite end of the first link. Since this other node must appear twice, it must belong to two links, the first link and another link. The script searches for the other link using the node ID and when it is found, saves it as the second link for the highway. The node at the opposite end of the second link is then used to search for another link that also contains this node. By continuing the process of matching the end node of one link to a node of the next link, a geographically ordered list of links for the highway is created.

Link records all contain start nodes and end nodes, but there is no sign convention or rule for deciding which node to call the start and which to call the end node. For this reason it often happens that the end node of one link is also listed as the end node of the adjacent link. To assemble the two links into a logically ordered route, the beginning and end nodes of the adjacent link must sometimes be reversed to ensure that the end node of one link connects to the start node of the next link. This is done automatically by the script, but to keep track of links that have had their nodes reversed, a single character (either an "O" signifying original order or an "R" signifying reversed order) is added to every link record for the highway.

## 5.5 Adding a Highway to the Route

After the links are edited, geographically ordered and marked with either an “O” or an “R”, the user is asked if he wants to add this highway to his route. If the answer is yes, the following data are stored in a MATLAB **mat file**: the geographically ordered list of link ID numbers, a string array containing the link records, lists of the starting and ending node ID numbers for all links, and an array whose rows contain the longitude, latitude, and ID number for each node. The file is stored under the name **HWY-STATE.mat**, where HWY represents the highway name as input by the user, converted to upper case, and STATE is the two letter state abbreviation. For example, data for 1-40 in New Mexico would be stored in a file named **I40\_NM.mat**. The script maintains a list of all of the highway data file names for the individual highway files and stores it in the **hnames.mat** file. If errors were made or if the wrong highway was processed or for any reason whatever, the user can decline to add the highway to the route, in which case nothing is saved for that particular highway and processing continues as if the highway had never been input.

After the user decides whether or not to add a highway to the route, he is asked if he wishes to process another highway. If the answer is “yes, ” he is again prompted to enter the state and highway designations just as he did for the first highway. The process described above is then repeated, except that there is no prompt about the use of a beginning stub. The user continues adding highways until he finally gets to the last highway on a route. Here special care must be taken to eliminate the links that continue on past the end of the route or the point at which the highway joins the end stub. Two menu items in the *Overlay* menu can be used to mark the location beyond which links must be eliminated. If the user selects *Mark End Stub*, a MATLAB file requester will appear and the user can select the end stub file. The script loads the end stub, finds the coordinates at which the end stub meets the highway, and plots a cross to mark that point on the plot. If no end stub is needed, the user can select *Mark a Location* which will bring up two input boxes for the user to enter the longitude and latitude where the route ends. With this information, the script plots a circle to mark the end point of the route.

## 5.6 Combining all the Highway Data for a Route

When the user declines to add another highway, he is given the option of continuing or terminating the script. If there have been some problems with the input, terminating here saves time and the individual highway link files are saved and available for inspection. However, usually the user will choose to continue, which causes the script to invoke the **common\_combo.m** function. This is where the individual highway links are pieced together to form a single geographically ordered list of links for the route. No user input is required by **common\_combo**.

The function begins by loading the **hnames.mat file**, which contains an ordered list of the file names for the highway link files. The first two highway files on the list are loaded and the function attempts to find the node at which the route leaves the first

highway and joins the second. To do this the lists of node ID's for both highways are stored as MATLAB vectors and each is sorted to put the node ID's in numerical order. This is done to simplify the task of finding the ID's of nodes common to both highways. If the highways have a simple intersection there will be only one common node, but if the highways meet and join together for some distance, a number of common nodes will be found. The function searches the geographically ordered list of links for the first highway to find the first link that has one of the common nodes for an end node. That link is used as the last link of the first highway and its end node is used as the common node for highways one and two. All of the first highway links up to and including the one identified as the last link are saved as part of the route and the remaining links are discarded. The common node ID is then used in a similar search to find the second highway link that has that node as a beginning node. This link is used as the first link for the second highway.

Links records for the part of the first highway belonging to the route are saved in the *rec* array, a list of the beginning node ID's for those links is saved in the *asave* vector, and the latitude, longitude, and node number of those nodes is saved in the *xsave* array. Then the data for the first highway is cleared from the function workspace and data from the third highway is loaded. Again, the node ID's are numerically ordered and the nodes common to the second and third highways are found. The last link of the second highway used by the route is found by again locating the earliest link of the second highway that has a common node for an end node. The first link of the third highway is the one having that same common node as a beginning node.

With the first and last link of the second highway identified, they and all of the links in between are appended to the *rec* array. Similarly, their beginning nodes are appended to *asave* and the nodal coordinates and ID's are appended to *xsave*. The data for the second highway is then cleared from the workspace and data for the fourth highway is loaded. The process of identifying the common node and first and last highway links is repeated over and over as the function processes data for all of the highways in the highways list. When finished, *rec* and *xsave* are saved to a file, *allinks.mat*, and data in *asave* and *xsave* are used to create a plot of links for the entire route. Figure 4 is an example of a plot for a complete route including an overlay of state boundaries. The user is prompted to press return after he has inspected the plot and is ready to continue. Once the plot has been dismissed, a prompt asks whether or not to continue.

## 5.7 Collecting Data and Creating the Route File

If the route appears satisfactory and the user chooses to continue, the *route-gen\_combo.m* function is invoked. It first prompts the user to provide a file name and a brief text description or message to be stored with the data. Then the process of grinding out a route file is begun by loading in *allinks.mat*, a file containing the complete list of geographically ordered links. The *hnames.mat* file is also loaded and its list of file names for the route highways is combined into a string variable, *highways*, that is saved along with the route data.

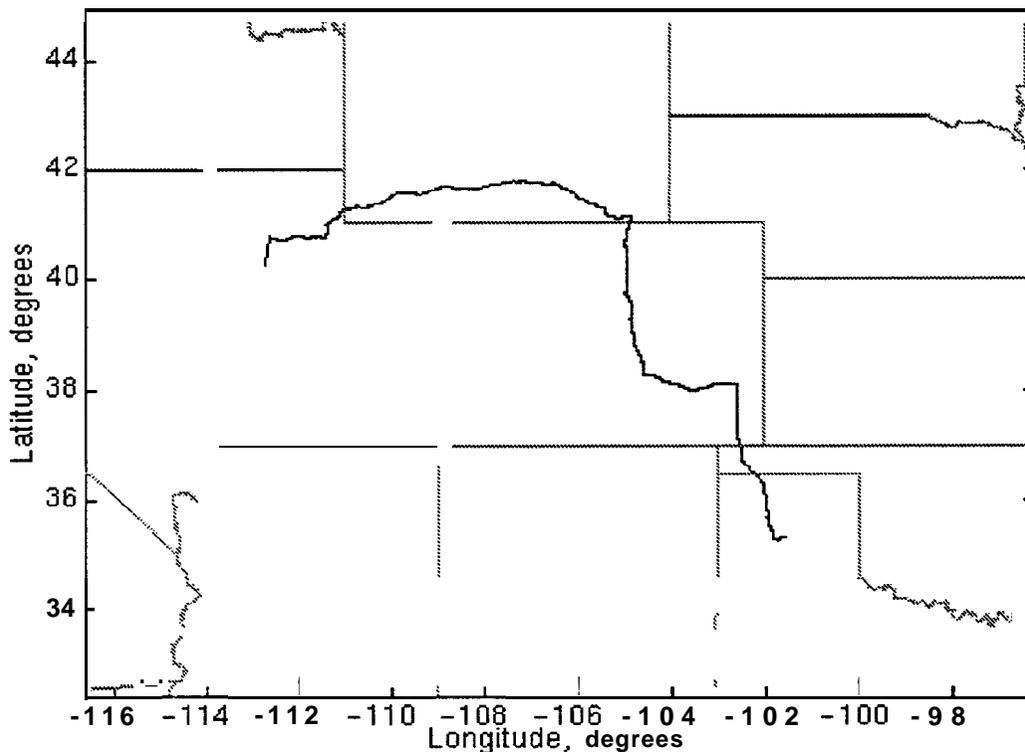


Figure 4: Plot of a Complete Route

Highway data collection begins by parsing the first link record to determine the state where the route begins and to gather the information needed to define *oe*, the operating environment array. Link records have three pieces of information that can be used to assign an operating environment. Character 111 of a link record contains an access control code, classifying the link as either full, partial, none or unknown access control. Character 117 classifies the link as either rural, small urban or large urban, and Characters 114-115 give the highway function class. Examples of function classes are Urban Interstate, Rural Interstate, Urban Principal Arterial, and Rural Local. In testing the script we have commonly found apparent contradictions between the three data fields. Often urban interstate function class highways are assigned the access control code for no access control. After consulting the Federal Highway Administration, it was decided to resolve any apparent inconsistencies between function class and access control code in favor of the function class. The script is coded to consider any highway whose function class is Interstate or Urban Freeway as limited access, no matter what access control is specified.

Meteorological stations are assigned to points on the route according to the map shown in Figure 5.



Each station is assigned a numeric code and every point of the route is assigned to a particular metstation by storing the metstation code in the *metsta* array. If a state has only one meteorological station, all links in the state are assigned to that station. If a state has more than one station, it is divided up and separate regions are assigned to each station. Some states have no stations and regions of the state are assigned to stations located in adjacent states. For example, western Iowa is assigned to Omaha, NE, and eastern Iowa is assigned to Peoria, IL. Most states have only one meteorological station, so after a link is parsed to identify the state, a meteorological station code can be assigned to all points for that link. If the link lies in a state that is divided into two or more regions, the individual coordinates for each point are examined to determine their correct meteorological station code. Metstation code assignments are given in Appendix C.

Geographical location information is obtained by loading in the MATLABized geo file for the state containing the first link. The ID of the first link on the route is used in a string matching search of the *geonew* array to find the record corresponding to that link. That record is then parsed to find the indices, in the *xgeo* array, of the coordinate pairs for the points belonging to the first link. The corresponding coordinates are extracted and used to define the first entries in the longitude, *lon*, and latitude, *lat*, arrays. However, before the points are saved, the link record is checked to see whether or not its end points had to be reversed to make the link orientation consistent with the route direction of travel. The order in which the coordinates are saved to the route coordinate arrays depends on whether or not the link orientation was reversed. The number of points saved to the coordinate arrays is noted and a similar number of entries in the operating environment and metstation arrays are generated. If the state has more than one met station, the individual coordinate pairs are examined before assigning the metstation codes.

When the function finishes dealing with the first link, the second link on the list is parsed to find its state and to collect the data used in assigning an operating environment code. If the state is different from the previous link, a new state geo file is loaded and processing continues as it did for the first link. Geo file data are used to find coordinates that are appended to the route coordinates array, and a corresponding number of metstation and operating environment entries are also appended to their respective arrays. This pattern repeats itself until all of the route links have been processed. At that point the only element of the route characterization file that is lacking is the route mile marker array *rmm*. Formulas from Reference [2] have been programmed into the function to calculate the distance between pairs of latitude/longitude coordinates. They are used to calculate the distance between successive points along the route and a running sum of the distances is saved as the route mile marker array.

The final step in the process is adding stubs at the beginning and/or end of the route file. A prompt requests the user to input whether or not any stubs are used for the route. If the user replies “yes,”) he is asked if a beginning stub is needed. If the answer is again “yes” the function checks to see whether or not the user selected a beginning stub while inputting the first highway. If he did, the user is asked to confirm that this is the correct beginning stub. If either this is not the correct stub or a beginning stub had never been selected, a file requester appears and the user is asked to choose a beginning stub.

The function checks to confirm that the end point of the stub matches the first point on the NHPN route. If it does, the stub is preappended to the route. If it doesn't, the stub is not used and a message to that effect is output to the screen.

After dealing with the beginning stub, the ending stub is handled in a similar way. The user is asked if he wants to use an end stub. If yes, the function checks to see if an end stub had previously been selected using the *Show End Stub* menu option. The user can confirm that this is the correct stub, or decline and select a different stub from a file requester. After a stub has been selected, it is checked to see if it matches the route end point. If so, it is appended to the route and if not it is not used and a message is printed to the screen. The *rmm* vector is then recalculated to account for the stubs and a congratulatory message is printed to the screen. The route file is saved under the name the user specified. It contains the 5 column vectors, *ion*, *lat*, *oe*, *metsta*, *rmm*, and two strings, *highways* and *descript*. As explained above, *highways* contains a list of all of the NHPN highways on the route, and *descript* is the user's brief text description of the route.

## 6 Comments and Excuses

Route\_Tool is very much the product of an evolutionary process. The basics of finding, parsing, and assembling the data were planned, but the major portion of the code involves editing and correcting the NHPN data, a task that was, for the most part, unanticipated. Problems were encountered when running early versions of the code and the various editing tools were added to handle them as they were discovered. Bifurcations and spurs were the earliest discoveries, but other problems quickly became apparent. For example, finding two distinct links for the same highway that have the same nodes was a complete surprise.<sup>4</sup> Perhaps highways without end points (loops) could have been anticipated, but they were not. Links marked as out of service and segments of Interstate highways with access control codes indicating unrestricted access also caused difficulties. As the need for data editing expanded, it quickly became apparent that a GUI approach was required. All of this development history is noted here as an excuse for why the basic code architecture is not as clean and tight as it could be. A simple schematic representation of the code is shown in Figure 6 to help the user understand the relationship between the scripts and functions used in the code. Numbered circles in the figure represent datafiles that are either read or written. Circle 4 represents an arbitrary number of individual highway data files, one is written for each highway on the route.

MATLAB may not be the ideal language for programming Route-Tool, but it is not without its good points. MATLAB is interpreted and has very good interactive capabilities, which helped very much in debugging the code. Also, the user may occasionally terminate Route-Tool prematurely and use the interactive mode to examine the environment (workspace in MATLABese). If so inclined, the user is encouraged to "hack around"

---

<sup>4</sup>Examples can be found by processing I10 in Arizona.

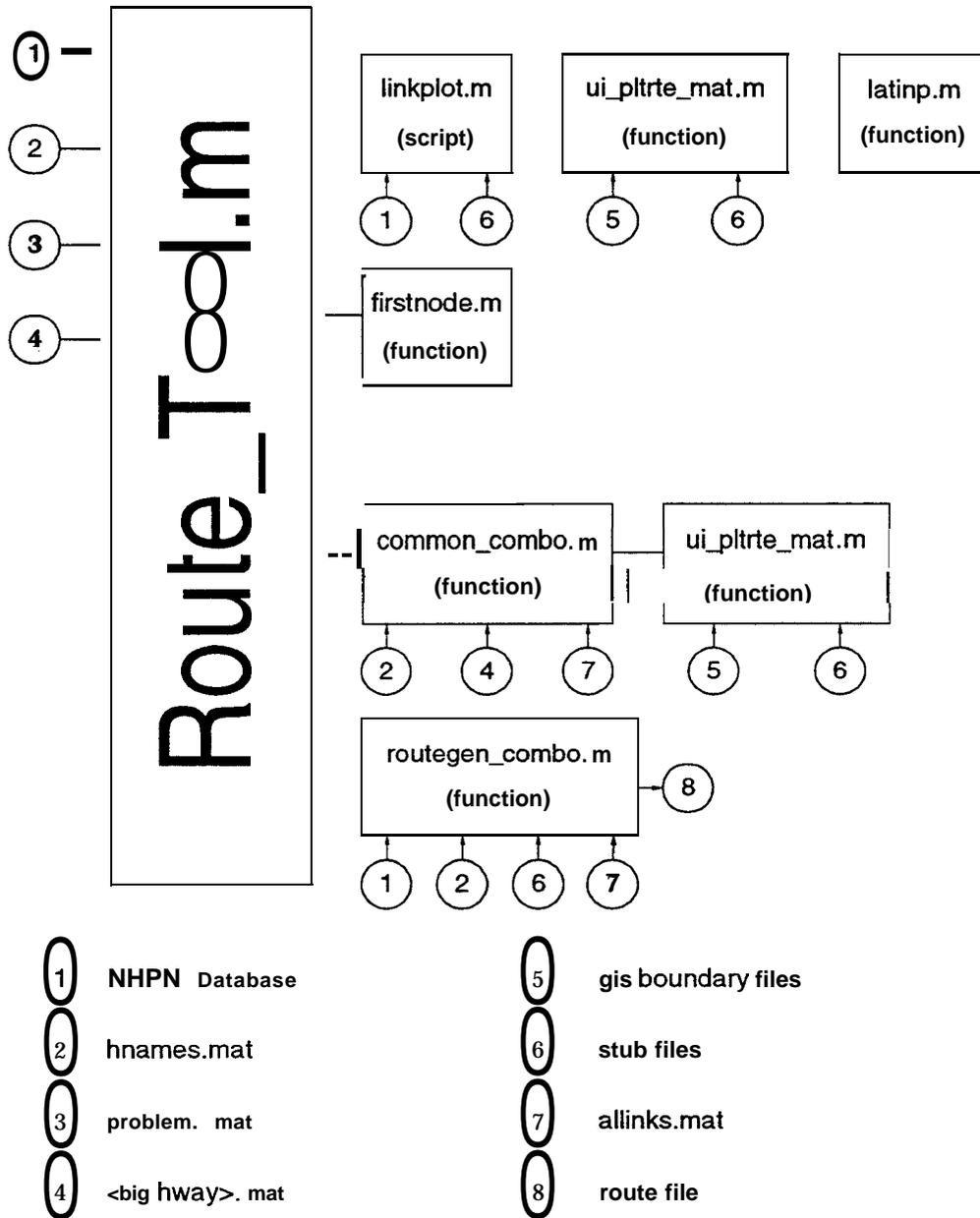


Figure 6: Code Architecture

with Route-Tool to help him understand the code and satisfy his curiosity. As a hackers' aid, a *keyboard* option was added to the GUI Options menu. Selecting *keyboard* breaks the execution and puts MATLAB in an interactive mode until the user enters return.

It will come as no surprise to anyone who reads the code that the author is not a computer scientist. Still, an effort was made to use structured programming and liberal commenting in the hope of making `Route_Tool` maintainable and updateable. That could be important if future releases of the NHPN use a data format different from the current version. The code was written to be reasonably user friendly and allow graceful recoveries if a problem occurs. Instructions and information are often displayed in the text window to make the code easier to use without having to consult this report. Inputs are checked and second chances are offered if they are found to be improper. At several places in the code the user is given the option of terminating rather than having to continue on with input known to be incorrect. Finally, the way the code is structured, it should be a simple matter to extend and include additional data in the route file if it ever becomes necessary. All of the parsing to collect and save information for the route file is confined to the `routegen_combo.m` function. Simply adding extra parsing and data saving statements in that script can extract additional data available in the link records.

## 7 References

### References

- [1] J. S. Phillips, "Route Characterization Using 1990 Census Data for use in Transportation Risk Assessments (U)", SAND 93-01 10, Sandia National Laboratories, Albuquerque, NM, April, 1994, The report is Classified.
- [2] J. P. Snyder, "Map Projections Used by the U.S. Geological Survey," , U.S. Government Printing Office, 1982, pp. 68-69.

## A Matlabizing the NHPN Database

Many of the operations used in creating a route file involve loading NHPN database files and performing string matching and search operations with the data. To speed up these processes it is convenient to store the link and node files as MATLAB **mat** files. A simple MATLAB script, `nhpn2mat-links.m`, was written to read in an NHPN link file, store all of its link records in a single large array variable, *link*, and save the array as a MATLAB file. Each row of *link* is a link record, i.e., a 121 character string, and *link* has as many rows as the state has links. The script repeats itself to automatically process all of the link files in the NHPN. Another script, `nhpn2mat_node.m`, does essentially the same thing for node files except that the array variable is called *node* and each of its rows is a node record, i.e., an 83-character string. This script also sorts the node array based on node ID number so the node records are stored in order based on node ID.

Converting the NHPN geo files is a more complicated matter because the process involves significant reorganization to speed data retrieval. The original NHPN geo files consist of blocks or groups of records, each group corresponding to a highway link. The first record in a group contains the link ID, state and county FIPS codes, ID's of the nodes belonging to the link, and the number of coordinate pairs used to define the link. The remaining records in the group consist of longitude/latitude coordinate pairs, stored four per record. Because the number of coordinate pairs varies from link-to-link, the number of records in a group also varies.

The `nhpn2mat_geo.m` script reads in the first geo record and parses it to find the number of coordinate pairs describing the first link. The first 33 characters of the initial geo record are stored in the array *geonew*. Appended to those are a six character representation of the number one and a six character representation of the number of coordinate pairs for the first link. Thus, the first row of *geonew* is a 45 character string. The records containing the coordinate pairs for the first link are read and each pair is stored as a row in an array, *xgeo*. That is, *xgeo* has two columns, the first for longitude and the second for latitude.

Next the script reads in the first of the second group of geo records. That record is parsed and a second 45 character string is saved as the second row in the *geonew* array. In this row the two, six character number representations at the end of the string give the number of coordinates for the first link plus one and the number of coordinates for the first two links. The records containing the coordinate pairs for the second link are then read and the coordinate pairs are appended as additional rows in the *xgeo* array. This same process is repeated over and over as the script reads through the complete geo file. For each group of geo records, i.e., each link, a single row is added to the *geonew* array and multiple rows of coordinate pairs are appended to the *xgeo* array. At the end of each line in *geonew* are two numbers that give the starting and ending *xgeo* row locations for the longitude/latitude coordinate pairs describing the corresponding link.

## B Creating an End Stub

If the first or last highway on a route is not in the NHPN database an end stub file must be created. Stub files are very short route files that can be appended to either end of a route. Stub files must obey two special rules: their beginning point must be the end of the route and their ending point must be a node on an NHPN highway. This requires that if the stub highway intersects the NHPN highway at a point other than a node, the stub must include the part of the NHPN highway between the intersection and the first NHPN node that the route passes over. Stub files can be created in two ways. For very short and simple stubs, a little knowledge about the highway and MATLAB can be all that are needed. For more complex stubs, a MATLAB script, `Stub_maker1.m`, can display a digitized map of the stub and allow the user to point and click coordinates for a stub. However, be warned that “point and click” is not intended to imply no work is required.

To use the `Stub_maker1.m` script, the user must obtain a digitized map, in “.bmp” format, that includes the complete stub. A convenient source of such maps is the DeLorme PC program `Map ExperM`. It allows the user to scale, annotate, and save maps in the “.bmp” bitmapped format. In addition, it provides a convenient way for the user to find the latitude and longitude coordinates at any points of interest. As saved, the map must include two reference points for which the user knows the latitude and longitude. These reference points are used to scale the map image so that the bitmap coordinates can be translated into longitude and latitude. Also, the coordinates of the intersection point between the stub and the NHPN highway must be known.

The script begins by displaying a MATLAB file requester and asks the user to select the map file to be used in creating the script. After the map is loaded and displayed, the user is prompted to input the longitude and latitude coordinates for one of the map reference points and then to click on the point. He is then prompted to do the same for the second reference point. The user should be able to locate the stub starting point on the displayed map, but he probably needs some help to identify the stub ending point. Recall that the ending point must be a node on the NHPN highway, but node locations are not always obvious and need to be marked.

A slightly modified version of the first part of `Route-Tool` is used to find the node locations of the NHPN link containing the stub intersection. The user is prompted to enter the state and NHPN highway name. That information is used to load in and parse the required data files and to open a window displaying a plot of the highway links. The plot will look like Figure 1, but without the stars marking the problem nodes. In addition, `Stub_maker1` prompts the user for the coordinates of the intersection point between the stub highway and the NHPN highway. Those coordinates are used to plot a large red X at the intersection of the stub and the highway. The user is then prompted to select the link containing the intersection point by placing the cursor on the link and clicking. The selected link changes color and can be unselected by clicking on it again. When the user is satisfied that he has chosen the correct link, the script saves the coordinates of its end nodes and deletes the link display window. The node coordinates are then used to plot

the end nodes on the imported map as white stars.

The script prompts the user to move the cursor into the map window and define a set of points for the stub by clicking on them, beginning at the end point of the route and moving toward the intersection with the NHPN highway. He is advised to stop adding points when the road changes operating environment (e.g., goes from no access control to limited access) or when the end point of the stub has been reached. In either case he presses return while the cursor is in the graphics window to quit entering points. A prompt in the text window then asks the user to supply the meteorological station code for the stub and the operating environment code. When those are supplied, he is asked whether or not he wishes to continue adding points. More points would be needed in cases where the operating environment changes before the end of the stub is reached. If he chooses to add more points, the map is redrawn with circles marking the points that have already been selected. The user then picks up where he left off selecting points for the stub. When finished he again specifies the meteorological station, the operating environment and indicates whether or not he wishes to select more points. Upon completion of the selection process he is prompted to provide a descriptive filename for the stub and to give a one line description of the stub.

For simple stubs, a non-graphical approach can be used. The first step is to locate data for the link of the NHPN highway that the stub highway intersects. This can be done in several ways, but one of the most straightforward is to use Route-Tool to create a temporary route that consists of that single link. Using the GUI *Mark a Point* and *Region Delete* options makes this task reasonably easy. After the route file has been created, MATLAB can be used in its interactive mode to make the stub file. Begin by loading in the data from the single link route just created into MATLAB's workspace. Inspect the coordinates of the points to find out where the link intersects the stub. For example, suppose the link has 17 points and the stub intersects it between the sixth and seventh points. Also suppose that the user wants to use three points to define the stub before it intersects the NHPN highway and that the stub ends at the beginning of the link. The coordinates for the stub are defined with the following two MATLAB lines:

```
>> lats= [xxxxxxx xxxxxxx xxxxxxx lat (6: -1: 1)] ;  
>> lons= [xxxxxxx xxxxxxx xxxxxxx lon (6: -1: 1)] ;
```

where the x's indicate user numeric input for the latitude and longitude of the three points not on the NHPN highway. Note that data variable names used for a stub file are the same as those for a regular route file with an "s" added to the end. This is done so that stub data and route data can coexist in the MATLAB workspace. The operating environment (assumed rural, no access control) and *metstation* are defined with the lines,

```
>> oes=[4 4 4 oe(6:-1:1)];  
>> metstas=metsta(1) * ones(1,9);
```

and the stub route mile marker vector is created from the coordinates by using the *ll2rmm.m* utility that should be included with the Route-Tool software;

```
>> rmms=112rmm (lons,lats);
```

The only remaining variables that need to be defined for a standard stub file are the end point coordinates and the description:

```
>> latends=lats(9);
>> lonends=lons(9);
>> descript='This is an example stub for the report.';
```

Had the stub turned toward the other end of the NHPN link after intersecting with it, the(6:-1:l)'s and (9)'s shown above would have been replaced with (7:17)'s and (14)'s. All that remains is to choose a file name for the stub file and save the variables, *lats*, *ions*, *oes*, *metstas*, *rmms*, *latends*, *lonends*, and *descript* under that name as a "mat" file. Since Route-Tool expects the user to be able to select a stub using a file requester, descriptive names should be chosen for stub files. For example, a stub that comes out of the Pantex plant, follows Farm to Market road 2373 to 1-40 and goes east on 1-40 until it encounters a node, might be named PTX\_FM2373\_140\_E. mat.

## C Assigning Meteorological Station Codes

Reference [1] chose fifty-one upper air meteorological stations for use in defining the statistical distribution of weather conditions in the vicinity of any point on a route. The country was divided into regions, each of which was assigned to a meteorological station and all highway links within a region are assigned to the meteorological station for that region. The regions and their corresponding meteorological stations and station codes are given in Table C below and are shown as Figure 5 in the text. The assignments are essentially the same as those used in Reference [1].

Region	Metstation Code	Metstation
Alabama	45	Centerville, Al
Arizona latitude >34.177	2	Winslow
Arizona latitude <34.177	35	Tuscon
Arkansas	51	Little Rock
California latitude >36.717	12	Oakland
California latitude <36.717	14	San Diego
Colorado longitude >-107	4	Denver
Colorado longitude <-107	36	Grand Junction
Florida latitude >29.7833	17	Apalachicola
Florida lat <29.7833 lon >-81.9	25	Cape Canaveral
Florida lat <29.7833 lon <-81.9	26	Tampa

Table 1: Meteorological Station Codes

Region	Metstation Code	Metstation
Georgia latitude >32.98	37	Athens
Georgia latitude <32.98	18	Waycross
Idaho	27	Boise
Illinois latitude >39.95	38	Peoria
Illinois latitude <39.95	8	Salem
Indiana longitude >-86.3333	9	Fairborn, OH
Indiana longitude <-86.3333	38	Peoria, IL
Iowa longitude >-93.5	38	Peoria, IL
Iowa longitude <-93.5	47	Omaha, NE
Kansas longitude >-97.5	20	Topeka
Kansas longitude <-97.5	39	Dodge City
Kentucky longitude >-86.333	9	Fairborn, OH
Kentucky longitude <-86.333	44	Huntington, WV
Louisiana	40	Lake Charles
Maryland	13	Sterling VA
Michigan latitude >44.167	33	Sault Ste. Marie
Michigan latitude <44.167	23	Flint
Minnesota	31	St. Cloud
Mississippi	16	Jackson
Missouri	7	Monett
Montana	34	Bismark
Nebraska	47	Omaha
Nevada latitude >39.167	6	Winnemucca
Nevada latitude <39.167	3	Desert Rock
New Mexico	1	Albuquerque
New York	42	Albany
North Carolina	43	Greensboro
Ohio	9	Fairborn
Oklahoma	49	Oklahoma City
Oregon	28	Salem
South Carolina	11	Charleston
South Dakota longitude >-100.5	21	Huron
South Dakota longitude <-100.5	24	Rapid City
Tennessee	50	Nashville
Texas longitude >-98.8	15	Longview
Texas lon <-98.8 lat >33.75	48	Amarillo
Texas lon <-98.8 lat <33.75	19	Lubbock
Utah	5	Salt Lake City
Virginia	13	Sterling
Washington longitude >-120.5	29	Spokane
Washington longitude <-120.5	30	Quillayute
Wisconsin	32	Green Bay
Wyoming	46	Lander

Table 1: Meteorological Station Codes, continued

Reference [I] defined metstations only for states, shown shaded in Figure 5, that had routes of interest for early ADROIT analyses. Should future requirements include highway sin Maine, New Hampshire, Vermont, Massachusetts, West Virginia, Rhode Island, Delaware, or New Jersey, meteorological stations will have to be assigned to those states and Route-Tool must be modified to incorporate the additions.

## 8 Distribution List

1	DOE/AL	S. B. Fellows TSD/AB
1	DOE/AL	S. A. Thompson
1	MS 0405	D. D. Carlson
1	0507	K. G. McCaughey
1	0557	T. J. Baca
1	0557	M. J. Sagartz
1	0661	E. D. Russell
1	0718	R. H. Yoshimura
6	0763	B. D. Boughton
1	0763	E. R. Copus
1	0763	D. B. Clauss
1	0763	T. D. Monina-Horn
1	0763	E. E. Ryder
1	0763	R. K. Wilson
1	0766	D. M. Ellis
1	0783	E. K. Lemon
1	9018	Central Technical Files, 8940-2
5	0899	Technical Library, 4916
2	0619	Review and Approval Desk, 12690 For DOE/OSTI