Integrating *Clarus* Weather Station Data and State Crash Data into a Travel Decision Support Tool

www.its.dot.gov/index.htm
Final Report — September 23, 2011
FHWA-JPO-11-162

U.S. Department of Transportation
Federal Highway Administration
Research and Innovative Technology Administration

Produced under the Road Weather Management Program
Federal Highway Administration and the
ITS Joint Program Office
Research and Innovative Technology Administration
U.S. Department of Transportation

Notice

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA-JPO-11-162		

4. Title and Subtitle	5. Report Date
Integrating *Clarus* Weather Station Data and State Crash Data Into a Travel Decision Support Tool	September 23, 2011
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
John D Hill, Colin Brooks, Tyler Erickson, Ben Kazoil, K Arthur Endsley	

9. Performing Organization Name And Address	10. Work Unit No. (TRAIS)
Michigan Tech Transportation Institute and Michigan Tech Research Institute 1400 Townsend Dr Houghton, MI 49931	11. Contract or Grant No.

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
ITS-Joint Program Office US DOT / Federal Highway Administration 1200 New Jersey Ave SE Washington, DC 20590	Final Report 9/24/2010-9/23/2011
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

2009 crash data from the State of Michigan was combined with weather data from four Clarus weather stations in the Upper Peninsula of Michigan. Crashes were monitored within a 50 mile radius and associated with weather conditions at the Clarus station. From this data, a series of regression models were then created based on critical tipping points of weather data, as well as continuous weather observations. This provides an algorithm consisting of seven risk equations which are used under differing weather conditions. The crash risk algorithm was then combined with a time based algorithm in order to recommend a route. The relative weighting of crash risk and time is established by the user. Using an open source geospatial routing tool and open source road network software, a recommended route is defined.

17. Key Words	18. Distribution Statement
Crash Risk, Regression, Decision Support Tool, Open Source, Road Weather Management, Road Weather Information System	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified		22	

Acknowledgements

The research team wishes to acknowledge the Road Weather Management Program, and in particular, Paul Pisano for technical advising and feedback throughout the project. Also, thank you to Jonathan Salzman who assisted on completion of the statistical analysis in the development of the crash risk algorithm.

Table of Contents

List of Tables

List of Figures

Executive Summary

2009 crash data from the State of Michigan was combined with weather data from four Clarus weather stations in the Upper Peninsula of Michigan. Crashes occurring within 50 miles of a weather station were identified and paired with the associated weather during that hour of time. Additionally, weather conditions during hours in which there were no crashes were also recorded. From this data, weather conditions which were significantly associated with increased crash risk were identified. A series of regression models were then created based on critical tipping points of weather data, as well as continuous weather observations. This provides an algorithm consisting of seven risk equations which are used under differing weather conditions.

The crash risk algorithm was then combined with a time based algorithm in order to recommend a route. The relative weighting of crash risk and time is established by the user. Using an open source geospatial routing tool and open source road network software, a recommended route is defined. The server framework is also open source and building a routing query for the database to execute is a thin convenience wrapper for the Crash Risk Aversion algorithm and largely consists of procedural data formatting, security, and abstraction. The database is still completely decoupled from the server and there are several alternative server frameworks that might be used to manage the database. Additionally, the development of the tool allows for use on a variety of web-browsing applications.

Identifying Crash Risk

Data Set Development

The first objective was the creation of a dataset which would contain data from both the Clarus weather stations in Michigan and Michigan crash data. Crashes from 2009 occurring within 50 miles of each of the 13 Clarus weather stations were identified. Simultaneously, the Clarus weather data was downloaded for 2009. Since Clarus weather stations have come online gradually, only data for four of the Michigan stations were available for 2009. This data was collected for every hour, on the hour. Only data which passed Clarus quality checks were included in this data set.

Once the crash and weather data were obtained, the two were merged based on time and location. Two of the Clarus stations were closer than 100 miles from each other, and as a result some crashes which occurred between the stations were within 50 miles of both. In this case, crashes were associated with the station to which they were closest. In the rare case of multiple crashes occurring within a one hour observation near a particular Clarus station, multiple observations were included in the data set for that hour at that station.

Each of 15 weather variables was evaluated to determine the degree to which they were significantly correlated with crash likelihood. This was done using a set of logistic regression models where crash outcome (yes or no) was the dependent variable and the weather variable under consideration was the lone independent variable. Some of the weather variables were continuous (such as ambient temperature and atmospheric pressure), and others were categorical (such as precipitation type). Categorical variables were decomposed into a set of binary variables (such as snow, rain, etc for precipitation type).

The final data set contains 25,568 observations, of which 4,832 include a crash which occurred during the hour. Upon review of the crash data, only four of the Clarus weather stations were collecting data in 2009. As a result, only these four locations were included. Weather variables in the data set include average wind speed, wind gust speed, road ice percentage, precipitation intensity, precipitation type, atmospheric pressure, dew point temperature, precipitation rate, visibility, air temperature, road surface conditions, ice or water depth, and surface freeze temperature. Crash information includes variables related to crash time, weather, location, number of vehicles, lighting conditions, and injury levels.

Regression Modeling

Of the fifteen variables tested, thirteen were statistically significant. These results are shown in Table 1. For the categorical variables, each level was tested independently. For these variables, only some of the levels may have been significant predictors of crash likelihood.

Table 1: Parameter estimates associated with Clarus reported weather conditions

Variable	Units	Directionality Tested	Parameter Estimate	p-value	Odds of Crash	LCL	UCL
Avg Wind Speed	m/s	Higher Speed	0.0225	0.014	1.023	1.005	1.041
Ice Percent	%	Greater Percent	0.00279	0.0288	1.003	1	1.005
Precip Intensity 3	no prec	1	-0.1065	0.0052	0.808	0.696	0.938
Precip Intensity 6	heavy prec	1	0.2806	< 0.0001	1.753	1.335	2.301
Precip Type 3	None	1	-0.1108	0.0156	0.801	0.67	0.959
Precip Type 5	Snow	1	0.3884	< 0.0001	2.174	1.627	2.905
Atm Pressure	mbar	Higher Pressure	-0.00658	< 0.0001	0.993	0.992	0.994
Dew Pt Temp	deg C	Higher Temp	-0.0157	< 0.0001	0.984	0.981	0.987
Relative Humidity	%	Greater Percent	-0.00431	< 0.0001	0.996	0.994	0.998
Precip Rate	cm/hr	Higher Rate	0.1114	0.0478	1.118	1.001	1.248
Visibility	1000m	Greater Visability	-0.04	< 0.0001	0.961	0.951	0.97
Air Tempurature	deg C	Higher Temp	-0.0133	< 0.0001	0.987	0.984	0.99
Surface Status 2	Error	1	-0.3283	< 0.0001	0.519	0.448	0.601
Surface Status 3	Dry	1	0.0868	0.0004	1.19	1.081	1.309
Surface Status 7	Ice Warn	1	0.2791	0.0099	1.747	1.143	2.671
Surface Status 8	Ice Watch	1	0.1392	0.001	1.321	1.119	1.56
Surface Temp	deg C	Higher Temp	-0.00547	< 0.0001	0.995	0.992	0.997
Wind Gust Speed	m/s	Higher Speed	0.041	< 0.0001	1.042	1.036	1.048

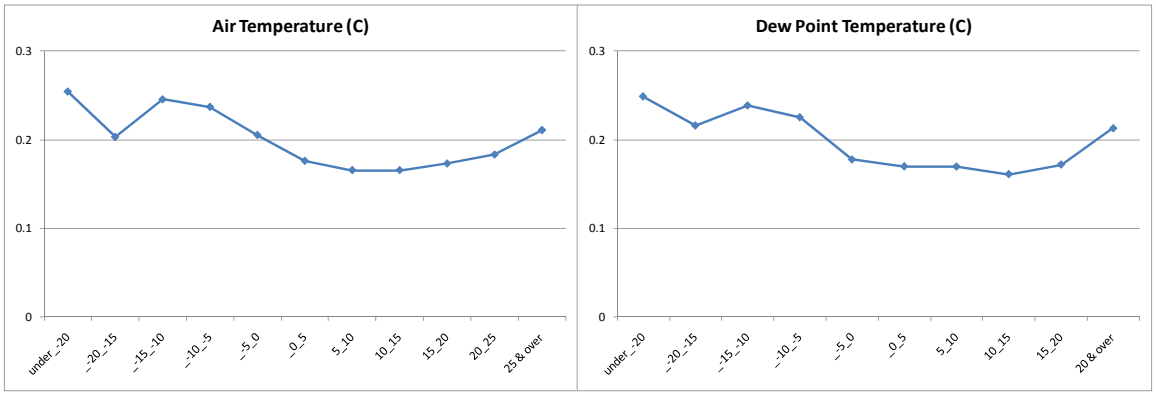
With regard to the categorical variables, both no precipitation and heavy precipitation were statistically significant in affecting crash risk, with crashes being less likely when precipitation was absent, and greater under conditions of heavy participation. Light and moderate precipitation did not significantly affect crash likelihood. The type of precipitation was also significant. No precipitation reduced crash likelihood, while snow increased crash likelihood. Rain had no significant effect. Some surface conditions were also associated with significant increases in crash risk. These included ice warnings and ice watches. Dry surfaces were also significantly associated with increased crash risk, although to a lesser degree. The only code for road surface that was significantly associated with reduced crash risk was an "error" code. Trace moisture, wet, chemically wet and frost were not associated with an increase in crash risk.

Among continuous weather variables, higher wind speeds were correlated with higher crash likelihood as was wind gust speed. Lower air temperatures, lower road surface temperatures and lower dew point temperatures were also associated with greater crash risk. Lower atmospheric pressure was also associated with increased crash likelihood as was decreased visibility and decreased relative humidity. Lastly, higher precipitation rate and greater percentage of ice on road were associated with higher crash likelihood. The only two variables not significantly related to crash likelihood were surface freeze temperature and surface ice or water depth.

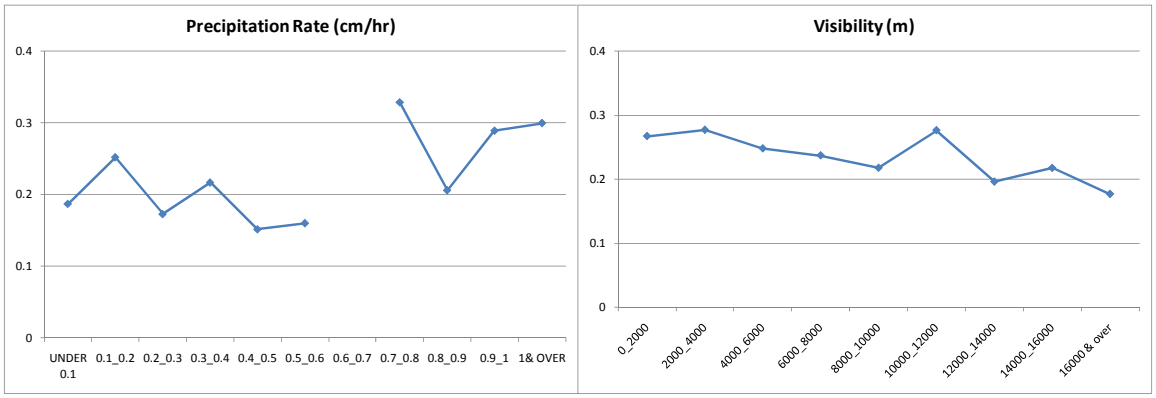
Cost functions for binary variables follow directly from the test of statistical significance. However, continuous variables may have non-linear relationships with regard to crash likelihood. In order to understand these potentially complicated relationships, distributions of crashes over the range of these continuous variables were developed. This will be used to determine how to evaluate the safety of road segments over the full range of weather measures.

Histograms showing the distributions of observations with and without crashes for each of the significant continuous variables are shown in figure 1. In some cases, as with Atmospheric Pressure

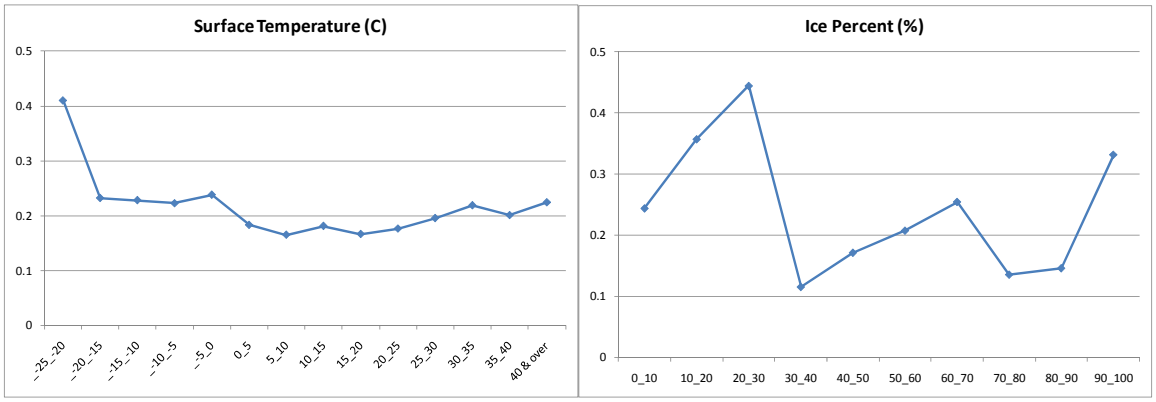
and wind speed, there seems to be a rather consistent relationship between changes in the weather variable and the change in the proportion of observations which have resulting crashes. In other cases such as ice percentage, there seem to be points of interest at which crashes drop off substantially before beginning another progression of increased risk. Other variables such as surface temperature and atmospheric pressure seem to show a particular narrow band of high crash likelihood that stands out from the rest of the data.



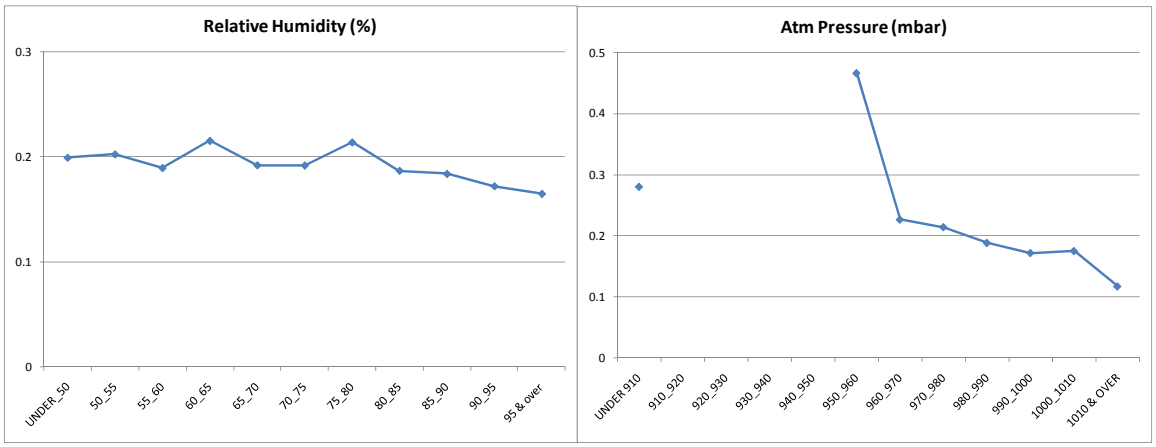
(a)



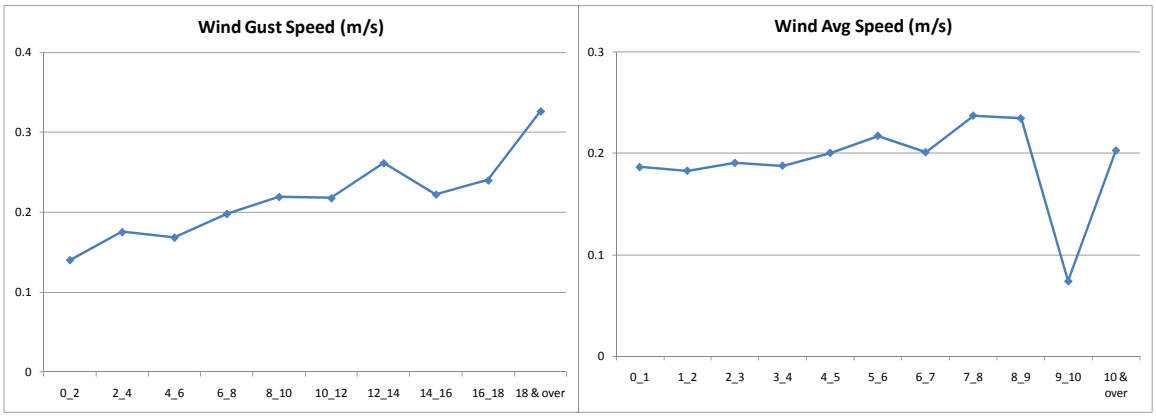
(b)



(c)



(d)



(e)

Figure 1: Distibution of significant variables and associated proportion of hours during which a crash occured

Decision Tree Structure

Upon identifying the points at which critical variables resulted in higher than average crash risk (such as temperatures below zero degrees C) a decision tree was created to identify the crash risk associated with the presence of these conditions. A series of logistic regression models were then created to determine how other weather factors influenced crash risk in these conditions. The resulting structure is shown in Figure 2.

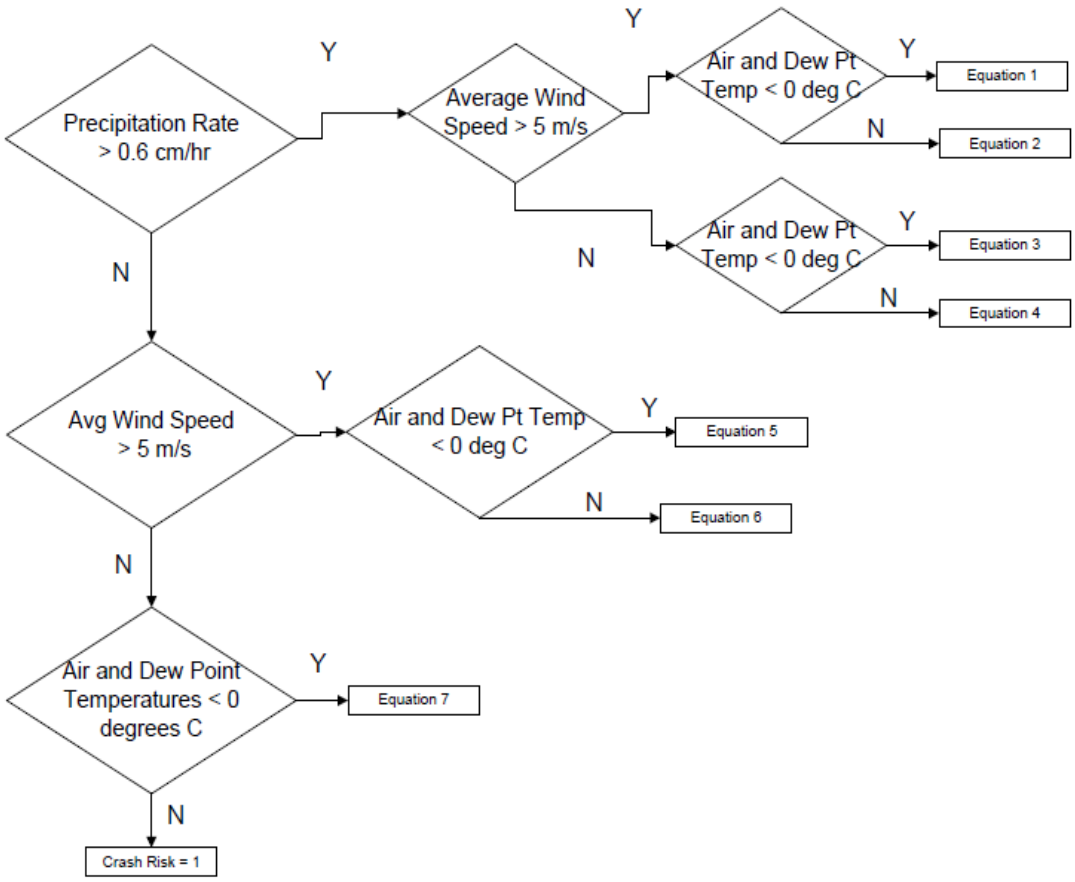


Figure 2: Decision tree structure identifying the conditions under which particular crash risk equations apply

The combination of a decision tree with the equations from the logistic regression model allow the algorithm to focus on the critical points beyond which crash risk increases and at the same time create equations which can predict crash risk over a continuous range of variables. Equations 1-7 as referenced in the above figure are shown below. Equations are relative to a baseline crash risk of 1, in which non of the critical air temperature, dew point temperature, precipitation rate or wind conditions are present.

Table 2: Equations for crash risk based on weather condition specific logistic regression models

Equation 1	Crash Risk = $e^{(0.6025+0.1716+0.2189)}$
Equation 2	Crash Risk = $e^{(0.6025+0.1716)}$
Equation 3	Crash Risk = $e^{(0.6025+0.2189-1.6789(\text{Air Temp})+1.4417(\text{Dew Pt Temp})}$
Equation 4	Crash Risk = $e^{(0.6025)}$
Equation 5	Crash Risk = $e^{(0.1716+0.2189)}$
Equation 6	Crash Risk = $e^{(0.1716-0.0245(\text{Dew Pt Temp}))}$
Equation 7	Crash Risk = $e^{(0.2189-0.0130(\text{Air Temp})+0.0438(\text{Avg Wind Speed}))}$

Temperatures are given in degrees Celsius. Wind speed is given in meters per second. The process by which crash risk is computed, is first to determine precipitation rate, wind speed, and air and dew point temperature. From there, the appropriate equation can be found on the decision tree depending on which side of the threshold these three variables or sets of variables are. It should be noted that air and dew point temperature must both satisfy the condition of less than 0 degrees Celsius in order for equations 1, 3, 5 and 7 to hold.

Algorithm Limitations

Typically, crash risk is in the context of the likelihood that an individual vehicle will be involved in a crash. This study analyzes crash risk from the perspective that at a given time, a crash will occur to some vehicle. The difference is subtle, but important. The data used for this study does not contain any sort of exposure metric such as vehicle traffic. This makes determining individual crash risk potentially problematic since traffic levels might drop off as road conditions worsen. As a result, it is important to note that any measure of crash likelihood represented in this model already accounts for the traffic levels (although unknown), for the weather conditons analyzed. In severe weather, the crash risk to an individual driver may actually be greater than shown in the model due to potential drops in exposure. However, there are some contexts in which overall crash likelihood is of interest. For example, road maintenance organizations are primarily concerned with overall safety of roadways, as opposed to incremental risk to a particular vehicle, when making road maintenance decisions.

Crash Risk Aversion Algorithm

Routing algorithms leverage network databases that internally segment roads into a discrete set of lines and nodes representing “streets” and “turns” (Figure 3). Road segments represent the multiple possible links that together may comprise a route. In order to connect these links, a routing algorithm needs to know which road segments connect and which directions those segments may be traversed. Along with these descriptors, other attributes stored in a network database include segment length and recommended travel speed. Combining the network attributes into a single traversal cost provides the input used by routing algorithms to optimize a least-cost path. “Crash risk” is an attribute stored similarly to length and speed and is accessed when calculating the total route cost. This chapter will describe the key elements of the algorithm and document its implementation in a web-based decision support tool.

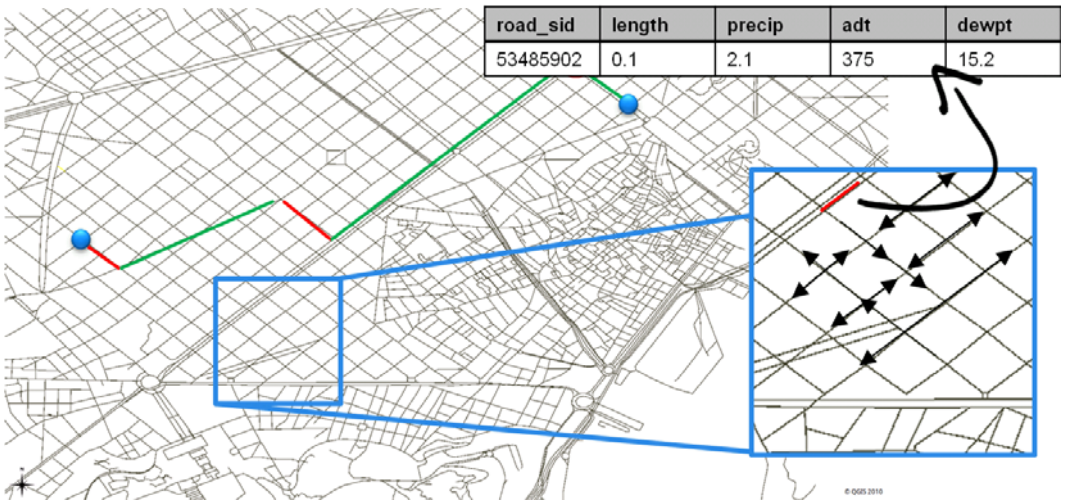


Figure 3: Example road network and its representation in a database as a table where each record corresponds to a road segment with various attributes.

The Crash Risk Aversion algorithm developed for this project is implemented as a collection of database objects that together are used to calculate a least-cost path between two coordinate pairs where the cost is a balance between travel time and crash risk. While typical routing algorithms use only the length or the travel time of a road segment to calculate the cost of a route, in this algorithm the cost depends on both travel time and crash risk, and allows users to adjust the significance of these two attributes in calculating a route.

Figure 4 summarizes the Crash Risk Aversion algorithm developed for this project. In the next few sections, the components of this workflow will be discussed in detail but here they are first summarized so to provide a high-level overview. First, the origin and destination coordinates are used to define a rectangular bounding box, buffered by half of a degree, and the road segments intersecting the buffered bounding box are considered for routing. A specified date and time for travel are used to

look up the weather at each Clarus station. In turn, these weather observations are interpolated for each of the selected road segments . With the interpolated weather at each road segment, the logistic regression decision tree calculates crash risk for each road segment. The user-selected risk modifier or α—which calibrates the relative importance of travel time vs. crash risk—adjusts the cost calculation to the level of risk aversion desired by the user. With the cost calculation complete, the problem becomes one of optimization. PgRouting, an open source geospatial routing tool was chosen for route optimization due to its tight integration with the project's database backend.

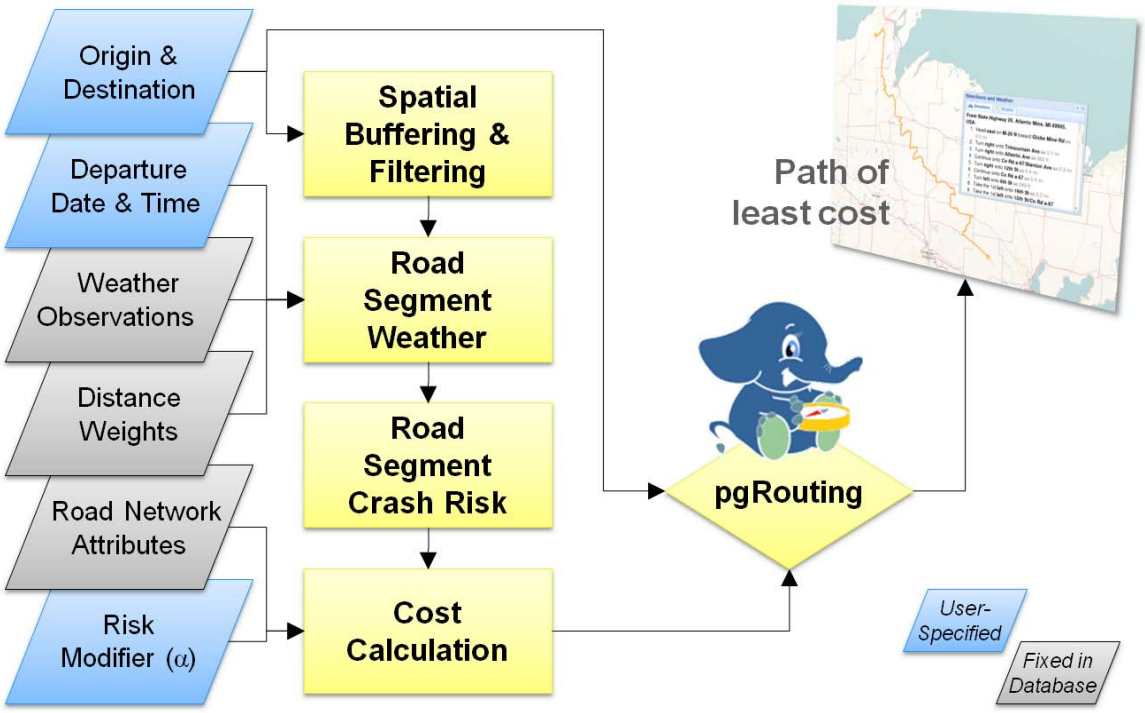


Figure 4: Flowchart summarizing the operations that make up the Crash Risk Aversion algorithm, where the final cost calculation is used by the routing library pgRouting to calculate a least-cost path.

Road Network Data Source

The road network for this study, for the state of Michigan, was downloaded by team from OpenStreetMap[2]. OSM is a crowd-sourced, freely downloadable dataset that is often viewed as a viable alternative to paid solutions (Hakley and Webber 2008). The downloaded data was checked for topological correctedness, and cost-related attributes such as traversal speed were verified before loading into the project database. Traversal speed was based on road type, with representative values for each major category, show below:

Table 3: Traverse speeds by road type

Road_Type	Speed (mph)
cycleway	1
footway	1
motorway	65
motorway_link	65
path	1
primary	55
primary_link	45
private	20
residential	25
secondary	34
secondary_link	34
service	15
steps	1
tertiary	45
track	1
trunk	65
unclassified	35
road	35
unsurfaced	0

Spatial Buffering and Filtering

As mentioned above, a buffer is applied to an origin-destination coordinate pair bounding box to limit the road segments comprising the set considered in the route calculation. Reducing road segment count lowers the computational overhead of the routing calculation. The spatial filter used to subset the road network data is a straightforward rectangular bounding box around the start and end points which is then buffered by half a degree. Buffering is necessary because road network geometry often demands travel in an initial direction away from the destination. Without buffering, this would take the route outside of the bounding box and violate the unbuffered spatial filter. Different buffer distances are available within the service. While the routing client at <http://geodjango.mtri.org/clarus/> uses a default buffer of 0.5 degrees, users can specify a different buffer on the interval 0.1 to 1.0 as a multiple of 0.1 by entering it into the URL. For example, to use the routing client with a buffer of 0.5 degrees, users can navigate to <http://geodjango.mtri.org/clarus/routing/0.5/>.

Road Segment Weather

The Crash Risk Aversion algorithm uses inverse distance weighting (IDW), given by Equation 1, as the weather data interpolation procedure. Weights are assigned for each road segment centroid (i.e. half the distance along a line segment) via IDW and later used in the cost function to estimate road segment-level weather conditions. While IDW is not the most rigorous spatial interpolation procedure,

the low sample count comprised of four weather stations is insufficient to calibrate more robust spatial covariance models required by methods such as kriging. The standard IDW equations to calculate an interpolated observation z are as follows:

$$z(r) = \sum_{i=1}^{N} \left(\frac{w_i(s) z_i}{\xi} \right)$$

Equation 1

where

$$\xi = \sum_{i=1}^{N} w_i$$

Equation 2

and

$$w_i(s) = \frac{1}{distance(r, s)^p}$$

Equation 3

A road segment center coordinate is represented by *r*, *s* is a weather station and corresponding spatial coordinates, *p* is the power parameter used to change the effect of distance (we use a power parameter of 2), w_i is the weight applied to a station observation z_i. The term ξ is introduced as a normalizing factor ensuring all weights sum to one.

The database stores the weighting factors *w* for each road segment and station combination, generating the interpolated weather value *z* in real-time. We decided to store the weights as opposed to the actual interpolated weather values for three reasons:

1. Storing values for each road segment and date-time combination results in an exponentially large number of records.
2. Weights are static as the distance between stations and road segments does not change. Storing the weights allows the system to expand as weather data is appended to the station records.
3. Adding additional stations requires only the re-calculation of weighting factors leaving the interpolation-to-estimated-weather relationship loosely coupled.

Crash Risk and Cost Calculation

The Crash Risk Aversion algorithm incorporates the concept of *crash risk* into the classical least cost problem, such that the optimal path is a balance of both travel time and safety (Equation 3). The user-defined parameter α, the risk modifier, is a weighting factor on the interval from zero to one; it can be used to adjust the importance of driving time versus crash risk in a routing query. The crash risk is calculated using the regression equations from Task 2 with the interpolated weather data at each road segment (Equation 4). All four stations are considered for each road segment, not just the nearest stations, where the distance was calculated from the road segment's midpoint to the weather station.

$$f(p) = cost_{p,t} = \alpha * traveltime_p + (1 - \alpha) * crashrisk_{p,t}$$

Equation 4

$$crashrisk_{p,t} = \frac{\sum_{s \in S} \lambda_s crashrisk_{s,t}}{\sum_{s \in S} \lambda_s}$$

Equation 5

pgRouting

With the addition of crash risk as a cost variable, total traversal costs for road segments are available. Selecting the least-cost path is an optimization problem for which multiple routing algorithms exist. The open source routing library pgRouting[1] extends PostGIS, an open source, spatially-enabled relational database builton top of PostgreSQL. In the database, tables containing geometric representations of the road network along with attributes describing traversal costs are stored and indexed for retrieval by SQL route-finding commands. The Crash Risk Aversion algorithm augments the built-in pgRouting Shortest Path Dijkstra[2] algorithm[3] by incorporating an additional cost metric in the function call. Other built-in routing algorithms could be similarly augmented.

The following subsections describe the database objects that are used to implement the crash risk algorithm.

Database Tables

Database tables store the raw information used to perform routing calculations.

ways - This table contains records that correspond to road segments. Relevant attributes include the geometry, length, start and end nodes, and road class. This table was modified by adding columns for weights used to interpolate CLARUS station weather data to the road segment, which were populated by Equation 1.

classes - This table contains records of distinct road classes. Relevant attributes include the nominal speed for driving on the segment. This tables was modified by adding a "speed" column, that stores the typical speed at which vehicles travel for this road class.

weather_station – This table contains records that correspond to CLARUS weather stations. Relevant attributes include the station code and point geometry (i.e. location).
weather_observation - This table contains records of CLARUS weather data measurements (air temperature, dewpoint temperature, average windspeed, precipition rate) for a single station at a single time.

[1] www.pgrouting.org
[2] http://www.pgrouting.org/docs/foss4g2008/ch07.html
[3] Other pgRouting algorithms are available in the software.

Database Functions

The database functions operate on the data in the database tables.

f_crash_risk(air_temp, dewpt_temp, avg_windsp, precip_rate)
This function calculates the crash risk index value for a given set of weather parameters. The function implements the logistic regression algorithm developed in Task 2.

f_routing_set(utc_time, alpha, x_min, y_min, x_max, y_max)
This function creates a set of road network segments that the routing algorithm operates on. To decrease computational cost, only a road segments that are fully or partially within the specified bounding box are considered. The routing set contains a cost attribute that incorporates both drivetime distance and crash risk, depending on the specified importance value (alpha). The crash risk is dependent on the time (and corresponding weather).

dijkstra_sp_delta_crashrisk()
dijkstra_sp_delta_directed_crashrisk()
These two functions generate an optimal route based on the Shortest Path Dijkstra routing algorithm, modified so that the cost of traversing network segments incorporates crash risk. The 'directed' version adds the ability to estimate optimal routes from directed networks (i.e. networks that include one way streets).

Discussion

The database functions were designed such that crash risk based routing can be done for any point in time for which weather data is loaded into the database. This allows a user to explore how routing changes over time, due to changes in weather. However, if routing calculations were only needed for a single point in time (such as the current time) much of the computation that is currently done in realtime could be precalculated in advance, resulting is dramatically faster routing calculations.

Decision Support Tool Development

This section describes how the database crash risk routing algorithms were made to be accessible over the web. As the Crash Risk Aversion algorithm was implemented in a PostgreSQL/PostGIS database, the server framework chosen for the web-based decision support would need an API consistent with the database server. The next few sections will detail the elements of this software stack and justification for the inclusion of each will be made. Throughout this chapter, the term "routing service" will be used to refer to the collection of server-side libraries and resources that respond to routing requests.

Server Framework

The chosen server framework, Django[4], is an open-source web framework written in the Python[5] programming language. It offers several salient features needed for this project including:

- A dynamic database API providing full abstraction of database objects and SQL commands in Python but still allowing complex raw SQL to be used where necessary
- Extensible template system
- Automatic and low-level caching

In the routing service, Django is running on top of the Apache HTTP web server, a popular web server framework (Netcraft, 2011). Apache relays HTTP requests and responses through the Web Server Gateway Interface (WSGI), a communications protocol for Python web applications. Apache is run on an Ubuntu Linux 10.04 LTS server for the purposes of this application.

Django's database API enables all objects in the database to be manipulated in Python including objects that weren't created through the API. This means that each database object has a Python representation. Tables are special Python classes called Models where each record in the table is an instance of the respective Model class. In the Django web framework, data models are exposed as resources on the web through a REST API. In summary, this means that all communication between the client and the server is stateless, cacheable, and HTTP-compliant. A RESTful service exposes internet resources to GET, POST (not acronyms) and other requests which can be made from any web browser. In the "routing service" created for this project, requests are initiated as HTTP GET requests for read-only resources. Such a request looks like like:

http://geodjango.mtri.org/clarus/route?method=CRA&startpoint=42.302 -83.689...

GET requests like the example above identify a resource (in this case the clarus/route resource) located in a specific domain (in this case, geodjango.mtri.org) and sometimes pass query parameters as key=value pairs which are listed after the question mark and separated by ampersands.

[4] www.djangoproject.com
[5] www.python.org

The RESTful interface used in the routing service allows us to define which HTTP request methods are permitted and what to do in each case, always giving an HTTP-compliant response. For this project, the routing service only responds to GET requests which customarily are for read-only resources. There is no reason to allow database content to be changed over the web for this project and so all other requests (i.e. POST, PUT, or DELETE) are ignored. This is a lightweight and reliable security measure that prevents any deletion, insertion or modification of database content through the web interface.

The routing service is represented by the workflow depicted in Figure 5. User data is transmitted by the client application as an HTTP GET request, received by the Apache server through a RESTful interface. These data are used to build a routing query which the server will execute the Crash Risk Aversion algorithm at the database level. Technically, the Apache server is making a request to the database server and returning the result to the client as an HTTP response. Building a routing query for the database to execute is a thin convenience wrapper for the Crash Risk Aversion algorithm and largely consists of procedural data formatting, security, and abstraction. The database is still completely decoupled from the server and there are several alternative server frameworks that might be used to manage the database. The bulk of this wrapper consists of finding the nearest "edges" or road segements to the user-specified start and end points. This is required as the coordinates of the start and end points do not necessarily fall directly on road segments. This database transaction, which returns the nearest road segments to the given coordinates, is a standard element of routing just as it is common practice to give directions from a person's house that start at the street, not the front door.

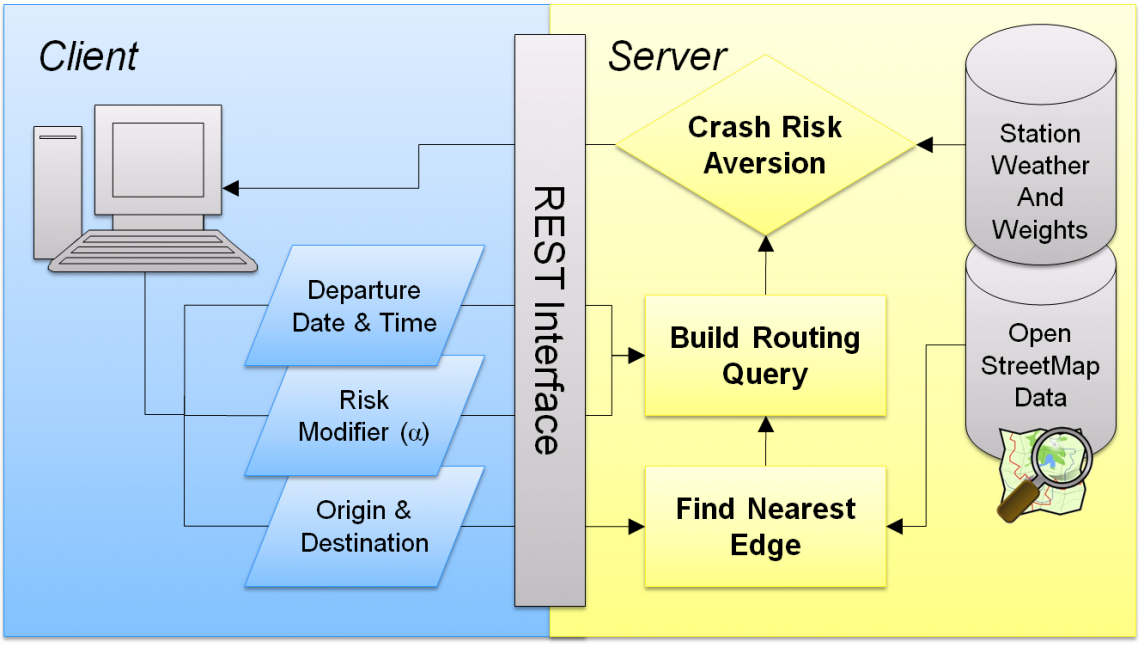


Figure 5: Client-server communication in the routing service is RESTful. There are two server-side procedures that operate on user data before the Crash Risk Aversion algorithm is run. They are part of a thin wrapper for database operations and implemented as part of the Django database API.

Client Application

Following the development practice of decoupling components, any client application capable of communicating through with the server through a RESTful API could have been selected for the user interface. The developers in this project again selected software libraries they are most familiar with and, in this case, were encouraged to do so as the pgRouting library included an extensible application already written in GeoExt, the library of choice. GeoExt[6] is an extension of the popular open-source user-interface library ExtJS[7]. With support for OpenLayers[8], the de facto standard for client-side web mapping interfaces, GeoExt enables rapid developing of web mapping applications.

GeoExt, ExtJS, and OpenLayers are all open-source libraries written in Javascript, the language of the web browser. As such, they offer cross-browser compatibility, consistent in both appearance and functionality. Javascript applications, unlike Java applets, Adobe Flash/Flex or Microsoft Silverlight applications do not require client-side libraries to be installed or updated. This makes Javascript the ideal choice for application development where the end users are primarily on government computers or other computing environments where the ability to install and update software libraries is compromised by IT security protocols.

The "components" of a Javascript application, at least where the term is used here, are the elements of the application which receive user focus and independently handle user interaction. They are enumerated differently depending on the level of granularity used in speaking about them. In the client application for the routing service (Figure 6), for example, we might say there are only two components: a top toolbar and a map panel. The top toolbar, however, might be described as a collection of six user input fields and buttons. However the components are defined, it is important to note that each of them has event listeners which describe behavior executed when certain events are fired and it is in this way the client application responds to user input.

[6] www.geoext.org
[7] www.sencha.com/products/extjs/
[8] www.openlayers.org

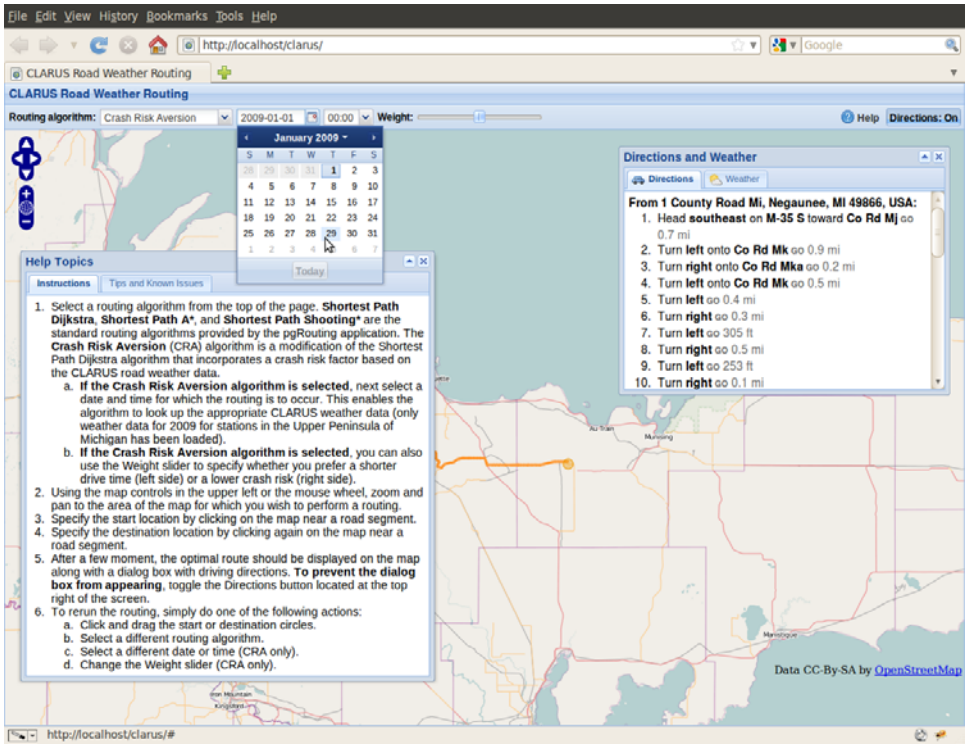


Figure 6: A screenshot of the routing web service's client application as it initially loads. Various components are displayed including the top toolbar, the map panel, a "date picker" calendar and two windows.

In the routing service's client application, the GeoExt library provides the components found in the top toolbar—a collection of user interface widgets that receive focus from the user, responding to single clicks, double clicks, dragging the mouse or other inputs. The date picker, the field containing the date selected for a routing query, responds to a user's click in the text field by receiving keyboard input and responds to a user's click on the icon to the right by showing a calendar from which the user can select a date. These events and the behavior they initiate are all contained within the date picker component and no other component is called or triggered. Contrary to that example, when a selection is made from the drop-down menu of routing algorithms (left-hand side of the top toolbar), the drop-down menu component fires events that affect other components. The map panel becomes masked (to intimate to the user that data are being loaded) and in the background a request to the server is made for the results of a routing query. When the routing data store (a client-side object in memory representing routes loaded from the server at any time) fires a "load" event to indicate it has successfully received a route from the server this event propagates throughout the application triggering other components. This is a short example of just one event chain in the web application.

The OpenLayers library provides the "slippy" map similar to the user-friendly feel of modern mapping web mapping services. It is analogous to the Google Maps API and uses the same technology to provide an experience similar to Google's popular mapping application. These applications provide the ability to map geospatial coordinates to pixel coordinates in a web browser. This enables a wide range of geospatial data to be displayed in virtually any (2D projection of a) coordinate system. Essentially, these libraries enable a 2D graph in any projection to be displayed but they are most commonly

employed for mapping the Earth's surface. In the routing service's client application, OpenLayers is used to capture user click events and the corresponding geographic coordinates. After the start and end coordinates for a route, along with other information captured in other components, is passed to the server and a response is received the part of that response that corresponds to the least-cost path between start and end points is plotted by the OpenLayers library. The path is stored as a vector layer object which means that later user input can modify this object and its corresponding representation on a map as in the case of moving the start or end coordinates.

The final piece of the client software stack is actually a remote service. The route layer displayed on the map, while a complete representation of the least-cost path provided by the algorithm, is not ideal for driving execution. In order to provide turn-by-turn directions, however, many services are needed to operate on route data. These services primarily include geocoding (translating spatial coordinates to addresses) and the translation of topology and directionality to English text. These services require large and complex database structures and algorithms which are outside of the scope of this project. There are currently no mature, complete open-source implementations of such a service. Consequently, the Google Maps API's directions service was leveraged for this purpose. The primary drawback of the Google Maps directions service is that it automatically calculates a route using Google's own algorithm, one where the only cost, presumably, is travel time. Such a route can be constrained, however, by user-defined waypoints (just as in the Google Maps application itself). While only 8 waypoints can be used to fit the route provided by Google to the route provided by the Crash Risk Aversion algorithm this seems to be adequate for most routes of short length or low crash-risk aversion (greater weight to the shortest path).

References

Haklay, M. and P. Weber. 2008. OpenStreetMap: User-Generated Street Maps. Pervasive Computing, IEEE. Vol. 7(4): 12-18.

Netcraft. "May 2011 Web Server Survey." <http://news.netcraft.com/archives/2011/05/02/may-2011-web-server-survey.html> Accessed September 19, 2011.

APPENDIX A. List of Acronyms

API	Application Programming Interface
AJAX	Asynchronous Javascript and XML
HTTP	HyperText Transfer Protocol
IDW	Inverse Distance Weighting
JSON	Javascript Object Notation
REST	Representational State Transfer
OSM	OpenStreetMap
SQL	Structured Query Language
WSGI	Web Server Gateway Interface
XHR	XML HTTP Request
XML	Extensible Markup Language

U.S. Department of Transportation
ITS Joint Program Office-HOIT
1200 New Jersey Avenue, SE
Washington, DC 20590

Toll-Free “Help Line” 866-367-7487
www.its.dot.gov

FHWA-JPO-11-162

U.S. Department of Transportation
Federal Highway Administration
Research and Innovative Technology Administration