Traffic Oracle

# Summary

Predicts traffic conditions for a given route on at a specific time and date based on historical traffic data

# Data Collection

## TomTom LiveTraffic Web Service

Traffic incident data can be collected from the TomTom LiveTraffic service at <http://www.tomtom.com/livetraffic>. This service has a web service interface which can be queried for real time traffic incidents for a square area defined by the geographical coordinates of the lower left and upper right corners. The live data is updated every 2 minutes.

The web service query is constructed as follows [[1]](#footnote-1)

http://www.tomtom.com/livetraffic/lbs/services/traffic/tm/1/{0},{1},{2},{3}/{4}/0,0,0,0/0/json/2bbdd0e2-6452-494a-b6b6-5aceb39048eb;projection=EPSG900913;language=en;style=s3;expandCluster=true

Parameters:

{0} Lower left corner latitude

{1} Lower left corner longitude

{2} Upper right corner latitude

{3} Upper right corner longitude

{4} Zoom level; 7 offers the most detailed list of incidents

expandCluster At certain zoom levels, clusters of traffic incidents are grouped together into one item in the response. Specify expandCluster=true so that the individual incidents in the cluster are returned as well.

The coordinates must be in the EPSG900913 projection system.

The response is in the form of a JSON string:

{

"tm":

{

"@id": "", *// Current time as Unix timestamp*

"poi": *// Point of interest*

[

{

"id": "", *// Unique jam identifier*

"p": *// Coordinates of jam*

{

"x": 0.0, *// Longitude*

"y": 0.0 *// Latitude*

},

"ic": 0, *// Type: 1 = unknown, 3 = accident cleared, 6 = traffic jam, 7 = roadwork, 8 = accident, 9 = long-term roadwork, 13 = unknown*

"ty": 0, *// Severity: 0 = no delay, 1 = slow traffic, 2 = queuing traffic, 3 = stationary traffic, 4 = closed*

"cbl": *// Cluster's bottom-left*

{

"x": 0.0, *// Longitude*

"y": 0 *// Latitude*

},

"ctr": *// Cluster's top-right*

{

"x": 0.0, *// Longitude*

"y": 0 *// Latitude*

},

"cs": 0, *// Amount of clustered (sub) points*

"cpoi": [ {} ], *// Clustered (sub) points of interest*

"d": "", *// Description*

"f": "", *// From: Jam starting point*

"t": "", *// To: Jam ending point*

"r": "" *// Road name*

"l": 0, *// Length of delay in meters*

"dl": 0 *// Duration-length of delay in seconds*

"c": "" *// Cause of accident*

}

]

}

}

## Traffic Scraper

A C# project (TomTomDataScraper) is attached which queries the web service in the Toronto area every two minutes, parses the response, and stores the results into a SQLAnywhere database. This project can be used as an example to build scrapers for other traffic data services.

The project uses a MercatorProjection class to project WGS 84 coordinates (-80, 43 and -78, 45 as the boundary corners of the Toronto area) into EPSG900913, and submits the query to the web service using an HttpWebRequest object. The response JSON string is deserialized into POI objects, the POI class containing the member definitions matching the format of the JSON string; EPSG900913 coordinates in the response are converted to WGS 84 coordinates before storage. Since the response contains both POI clusters (lists of POIs) and single POIs in the "poi" list, the list is flattened by extracting the sub POIs from the "cpoi" member of clusters.

The final list of POIs is inserted into a SQLAnywhere database table with the following schema:

CREATE TABLE "dba"."Traffic" (

// Unique id

"id" BIGINT NOT NULL DEFAULT AUTOINCREMENT UNIQUE,

// Timestamp from the web service response

"request\_time" "datetime" NOT NULL,

// Incident ID

"jam\_id" VARCHAR(50) NOT NULL,

// Incident description

"description" LONG VARCHAR NULL,

// Type of the incident, see TomTom LiveTraffic Web Service for values

"jam\_type" INTEGER NULL,

// Severity of incident, see TomTom LiveTraffic Web Service for values

"severity" INTEGER NULL,

// Longitude and latitude of incident in WGS 84 coordinates

"latitude" DOUBLE NULL,

"longitude" DOUBLE NULL,

// Response sometimes contains the street names for the start and end points of the

// incident and the street it runs along

"starting" VARCHAR(50) NULL,

"ending" VARCHAR(50) NULL,

"road" VARCHAR(50) NULL,

// Length of the delay in meters

"delayLength" BIGINT NULL,

// Duration of the delay in seconds

"delayTime" BIGINT NULL,

// Usually empty

"cause" LONG VARCHAR NULL,

PRIMARY KEY ( "id" ASC )

) IN "SYSTEM";

# Server Code

The SQLAnywhere server with the traffic incident data is set up to expose a web service (getDelay) which expects a HTTP parameter containing a XML string describing the route as well as the target time and date to calculate delays for. Note that to enable web services in SQLAnywhere, the server must be launched with the "-xs" option to specify protocols the server will listen on. For example, "-xs HTTP(PORT=80)" allows the server to listen for HTTP requests on port 80.

When the server receives a request to calculate delays, a stored procedure is used to find all the traffic incidents in the database which the supplied route might pass through at the specified date and time. The incidents are then aggregated based on location, and the probability of each incident cluster affecting the specified route is calculated based on the correlation between the date and time specified in the request and the date and time the incidents occurred at. The incident durations are scaled by the probability of occurrence, and the complete incident data is sent back to the client.

The server-side SQL code can be found in "server procedures.sql".

## Server Side Incident Selection Algorithm

On the server, the list of incidents which relate to the input LineString steps of the route is selected from the incidents in the database as follows:

The innermost SELECT statement selects all incidents from the incident table which has a known incident type, joined to the srsid lookup table on the geographic coordinates of the incident; this adds the id of a Spatial Reference System which contains the incident location, into which the input LineString could be projected in order to accurately compute the distance between the LineString and the incident point. If an incident does not have a delay duration value, this SELECT also takes the average delay per metre for each incident severity over all incidents (cached in the traffic\_delay\_avg table) and multiplies the incident road length by the average delay as the total delay for the incident. Also, \_1 and \_2 are stripped from the end of incident IDs since those denote the same incident in different directions, and should not be considered separate incidents.

The result set above is left joined to the list of input step LineStrings on the condition that

1. The incidents occurred within a certain time frame, depending on the target time which is the inputted travel time plus the duration of the trip as of the current step:
   1. The hour of the incident must match the hour of the target time
   2. The minute value of the incident must be within 4 minutes of the target time
   3. For non-construction type incidents, then only match incidents which occurred on weekdays if the target time is a weekday and only match weekends if the target time is on a weekend
   4. For construction type incidents, then only match incidents which occurred on the same day or the day before the target time; however, if data is only available up to a date before the target time (e.g. travel date is August 31st but we only have data up to August 25th), then match incidents which occurred on the latest date available.
2. The input step LineString, projected into the SRS of each incident, is within the incident road length of the incident.

The following example illustrates the data so far:

Input steps:

|  |  |
| --- | --- |
| LineString | Duration (start time of the step) |
| StepA | 0 |
| StepB | StepA start time + time taken for StepA |

Incidents:

|  |  |
| --- | --- |
| Incident | Incident road length (m) |
| IncidentA\_1 | 5000 |
| IncidentA\_2 | 5000 |
| IncidentB | 3000 |
| IncidentC | 1000 |
| IncidentD | 500 |
| IncidentE | 2000 |

Say StepA is 3000m from IncidentA, 2000m from IncidentB, 2000m from IncidentC, 1000m from IncidentD, and 1000m from Incident E. StepB is 6000m from IncidentA, 500m from IncidentB, 800m from IncidentC, 2000m from IncidentD, and 2000m from IncidentE. Also assume all the incidents satisfy all the time limitations.

The result set from the JOIN looks like the following:

|  |  |  |
| --- | --- | --- |
| LineString (not actually in result set) | Incident | Other Incident Data… |
| StepA | IncidentA (originally IncidentA\_1) | … |
| StepA | IncidentA (originally IncidentA\_2) | … |
| StepA | IncidentB | … |
| StepA | IncidentE | … |
| StepB | IncidentB | … |
| StepB | IncidentC | … |

We now have a list of incidents which match the eligibility criteria for the list of input steps. However, these incidents are not unique, so a GROUP condition is added to the JOIN to group by the date of the incident, the incident ID, and the type of the incident. The final joined result looks like the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Incident | Incident Date | Incident Duration (s) | Other Incident Data… |
| IncidentA | 1 | 100 | … |
| IncidentB | 2 | 100 | … |
| IncidentC | 2 | 100 | … |
| IncidentE | 2 | 100 | … |

We now have a unique list of eligible incidents.

In the same SELECT, a weekday factor is calculated for each incident group based on the day of the week it occurred on:

* If the incident occurred on the same day of the week as the target date, the factor is 1.0 i.e. full impact.
* Otherwise if the incident occurred on a weekday and is a construction type incident, the factor is 5/7 since there is a 5 in 7 chance the same construction will occur again during the week. If the incident if not a construction type incident, then the factor is 1/5 since there is 1 in 5 chance the incident will occur on any given weekday.
* If the incident occurred on a weekend and is a construction type incident, the factor is 2/7 following the same logic as above. If it is not a construction type incident, the factor is 1/2.

The next SELECT groups incidents in the same day within a small area (currently set to longitude and latitude between two incidents within two decimal places of each other) together. Say incidents A, B, and C satisfy this criterion; B and C are grouped into one incident and their durations are summed, but A is not since it occurred on a different date.

The result is the following:

|  |  |  |  |
| --- | --- | --- | --- |
| Incident | Incident Date | Incident Duration (s) | Other Incident Data… |
| IncidentA | 1 | 100 | … |
| IncidentB + C | 2 | 200 | … |
| IncidentE | 2 | 100 | … |

The final SELECT groups all the incidents in a small area together, similar to the above step; however, in this step the date of the incident is not a group parameter, and therefore all incidents within an area are collected regardless of which day they occurred on.

Next, a chance for incidents to occur at a specific location is calculated based on the weekday factor and the incident type. If the incident is due to construction, the chance is simply the average weekday factor over the incident group. Otherwise, the average weekday factor is multiplied by the number of occurrences of the same type of incident within a small area over all days, which can be seen as the total number of days on which a certain area has had traffic incidents; this number is divided by the maximum such number over all incident clusters for the final chance of a traffic incident occurring in a specific area.

The duration of each incident group is the average duration over the group multiplied by the chance factor.

Continuing with our example from above, this step would produce:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Incident | Number of days where there was an incident in this area | Chance factor (number of days / max number of days) | Expected delay duration for this area | Other Incident Data… |
| IncidentA (on day 1) and IncidentB + C (on day 2) | 2 | 1.0 | 150 \* weekday factor | … |
| IncidentE | 1 | 0.5 | 100 \* 0.5 \* weekday factor | … |

# Client Application

The web application is written in HTML and JavaScript, and makes use of jQuery and the Google Maps JavaScript API for its route mapping functionalities. The code for the client application is in "index.html" and "js/calcDelay.js" located in the WebClient directory.

The HTML is split into two parts, the left side containing a div element which is used to host an instance of the Google Maps object, and the right side contains user controls as well as a display panel for the route information.

To calculate the delay for a route, enter values for the start and end points in the appropriate text boxes as if using Google Maps, and optionally specify the time and date for which the delay should be calculated. The start and end points will be used to query Google for a list of possible routes, and when the query returns successfully the routes will be shown on the map object as well as the information panel.

A successful route query response from Google contains a list of routes, each route containing a list of legs, each leg containing a list of steps, and each step containing a list of coordinates of the path which makes up the step. In the callback handler, the response data is serialized into XML and used as a parameter in the SQLAnywhere web service call with the following format:

<route>

<time>DateTime to calculate the delay for</time>

<leg>

<step>

<duration>Total duration travelled so far</duration>

<lineString>A lineString in the WKT format</lineString>

</step>

<step>

... remaining steps of the leg

</step>

</leg>

<leg>

... remaining legs of the route

</leg>

</route>

For each individual route, a separate web service call is made using HTTP POST; the response is in JSON and contains a list of possible traffic incidents for that route. When a successful response is received from the server, each incident is drawn to the map as a transparent circle centered at the incident location, with the length of the incident in meters as the radius of the circle. The colour and opacity of the circle depends on the chance of the incident occurring and the severity of the incident. Finally, the sum of delays from the incidents is added to the route's travel time and shown in the route information panel.

1. Source: <http://blog.bsmulders.com/2012/03/using-the-tomtom-live-traffic-api/> [↑](#footnote-ref-1)