# Select the best change station in 2 bus route trips using Data Visualization

Shaked Kaufmann

Data Visualization Course

Summer 2015

Contents

[Introduction 3](#_Toc429618477)

[Data Set Description 3](#_Toc429618478)

[User Task and Visualization Purposes 4](#_Toc429618479)

[Visual Design Iterations 4](#_Toc429618480)

[Iteration 1 4](#_Toc429618481)

[Data to Visual Attributes Mapping 4](#_Toc429618482)

[Code 4](#_Toc429618483)

[Results 9](#_Toc429618484)

[Advantages, Disadvantages & Alternatives 9](#_Toc429618485)

[Iteration 2 9](#_Toc429618486)

[Data to Visual Attributes Mapping 9](#_Toc429618487)

[Code 10](#_Toc429618488)

[Results 12](#_Toc429618489)

[Advantages, Disadvantages & Alternatives 12](#_Toc429618490)

[Evaluation 12](#_Toc429618491)

[Reflection on Design Decisions 12](#_Toc429618492)

[The Value of the Visualization 13](#_Toc429618493)

# Introduction

This tool aims to visualize public transportation scheduling information in order to allow users to easily choose between changing stations in a trip combined of more than 2 bus routes.

The data is real data obtained from the Israeli data.gov.il website.

This report and data visualization were prepared as part the course “Information Visualization” that took place in Ben Gurion University during the 2015 summer semester.

# Data Set Description

The data set is comprised of 3 CSV files which were compiled of a greater public transportation scheduling data set.

The scheduling data set includes all the needed information about public transportation schedules in Israel, including: routes, trip times, stations geo-location and names, routes detailed path description, and stop times at each station.

#### stopShapes52473\_171.csv and stopShapes61551\_501.csv

|  |  |  |  |
| --- | --- | --- | --- |
| SHAPE\_ID | SHAPE\_PT\_LAT | SHAPE\_PT\_LON | SHAPE\_PT\_SEQUENCE |
| 61551 | 32.05568 | 34.77978 | 1 |
| 61551 | 32.05601 | 34.77852 | 2 |

These files contain a detailed list of hundreds of geo-location shapes that together allow an accurate rendering of the routes’ paths over a map.

#### stopTimesBothLines501\_171

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | TRIP\_ID | ARRIVAL\_TIME | DEPARTURE  TIME | STOP\_ID | STOP\_CODE | STOP\_NAME | STOP\_LAT | STOP\_LON | WAIT  TIME | WAIT  RANK | IS  ORIGIN | IS  DEST | LINE  NUM | | 4633412\_080815 | 07:47:19 | 07:47:19 | 29310 | 20004 | המסגר/יד חרוצים | 32.06129 | 34.78481 | 00:05:19 | 1 |  |  | 171/501 | | 4633412\_080815 | 07:52:19 | 07:52:19 | 13204 | 21669 | קניון עזריאלי/דרך בגין | 32.07571 | 34.79162 | 00:08:19 | 2 |  |  | 171/501 | | 4633412\_080815 | 07:55:19 | 07:55:19 | 13354 | 24068 | מסוף 2000/דרך נמיר | 32.08364 | 34.7951 | 00:09:19 | 3 |  |  | 171/501 | | 4633412\_080815 | 07:56:19 | 07:56:19 | 13154 | 21564 | דרך נמיר/ז’בוטינסקי | 32.08637 | 34.7955 | 00:10:06 | 4 |  |  | 171/501 | | 4633412\_080815 | 07:58:19 | 07:58:19 | 13181 | 21615 | דרך נמיר/פנקס | 32.09201 | 34.79521 | 00:11:06 | 6 |  |  | 171/501 | | 4633412\_080815 | 08:01:34 | 08:01:34 | 29339 | 20050 | דרך נמיר/חיים לבנון | 32.1035 | 34.79275 | 00:11:21 | 7 |  |  | 171/501 | | 4633412\_080815 | 08:03:34 | 08:03:34 | 13760 | 25985 | קאנטרי ל/דרך נמיר | 32.10925 | 34.79213 | 00:12:54 | 8 |  |  | 171/501 | | 4633412\_080815 | 07:31:19 | 07:31:19 | 15564 | 31252 | דרך השבעה/המלאכה | 32.0218 | 34.80551 |  |  | TRUE |  | 171 | | 17529279\_080815 | 07:55:40 | 07:55:40 | 29385 | 20118 | מחלף גלילות מערב | 32.14144 | 34.80176 |  |  |  | TRUE | 501 | | 17529279\_080815 | 07:48:13 | 07:48:13 | 13067 | 21415 | דרך נמיר/יהודה המכבי | 32.09503 | 34.7945 |  |  |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |
| This table holds the actual data about the trip and possible change stations.  For every trip (instance of a bus route/line), the arrival & departure time are specified for each station. Into this table, data about station features (names & geo-location) were integrated. Finally, the waiting time was calculated for each change station, a rank was assigned and additional route data was added as shown above.  As mentioned above, the full data set is comprised of 18 million records is rawer format than the data presented above. However, the data manipulations included only waiting time calculation between the relevant pairs of stop times and waiting rank. |  |  |  |  |  |  |  |  |  |  |  |  |

# User Task and Visualization Purposes

The purpose of the visualization is allow users to easily find the best change station for a given trip that is combined of 2 or more bus routes.

The “best change station” is considered to be the station where the user is expected to wait the shortest time when changing bus routes on his trip. This definition is somewhat basic and can be extended in the future to include less-dense stations for example (this would require more data operations and aggregation).

The user task can therefore be described as follows: “Given a selected bus trip journey that is combined of 2 or more routes, the user is presented with the best stations to change between routes, these are the stations that are visually ranked by shortest waiting time”.

# Visual Design Iterations

## Iteration 1

### Data to Visual Attributes Mapping

Set of geo-location points representing a route’s path -> shape: colored line

Change station -> shape: circle

Change station ranking -> sequential circle colors **and** circle sizes

The coloring was chosen as follows:

3 categorical colors:

#e41a1c – line 171

#377eb8 – line 501

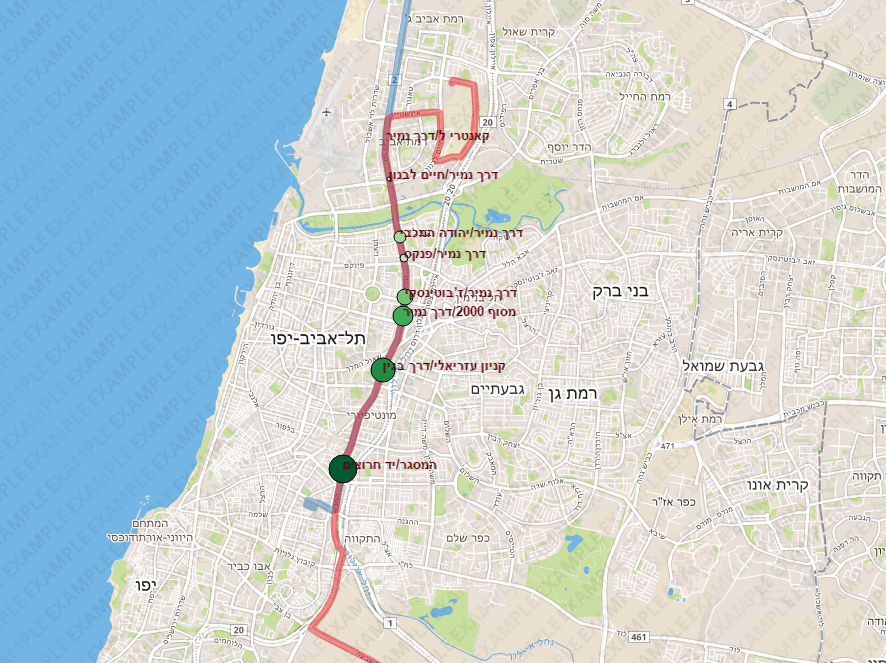
#4daf4a – base color to be used for a sequential coloring scale of the ranked stations. Up to top 9 possible change stations will be displayed. A higher number wouldn’t allow an efficient range where users can differentiate between the colors.

Change station ranking was mapped as follows: the higher the rank (i.e., the change station that has the shortest waiting time is ranked 1), the darker is the circle’s color and the longer is the radius.

### Code

<!DOCTYPE html>  
<html>  
*<!--import thr leaflet css-->*<link  
 rel="stylesheet"  
 href="http://cdn.leafletjs.com/leaflet-0.7/leaflet.css"  
 />  
  
<head>  
 *<!--Script imports-->* <meta charset="utf-8"/>  
 <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7/leaflet.css"/>  
 <script src="http://d3js.org/d3.v3.min.js" type="text/javascript"></script>  
 <script src="http://cdn.leafletjs.com/leaflet-0.7/leaflet.js"></script>  
 <script src='https://api.tiles.mapbox.com/mapbox.js/v1.6.4/mapbox.js'></script>  
 <link href='https://api.tiles.mapbox.com/mapbox.js/v1.6.4/mapbox.css' rel='stylesheet'/>  
 *<!--General Styling-->* <style>  
 html,  
 body {  
 height: 100%;  
 width: 100%;  
 margin: 0;  
 }  
  
 #map {  
 width: 100%;  
 height: 100%;  
 }  
  
 svg {  
 position: relative;  
 }  
  
 path {  
 fill-opacity: 0;  
 stroke-width: 7px;  
 stroke-opacity: 0.5;  
 }  
  
 .stationsText {  
 stroke: black;  
 stroke-width: 0.1px;  
 fill: #67000d;  
 font-size: 10pt;  
 font-weight: bold;  
 }  
  
 .stations {  
 stroke-width: 1px;  
 stroke: black;  
 }  
  
 .lineStations {  
 fill-opacity: 0;  
 stroke-opacity: 0;;  
 }  
  
 </style>  
  
</head>  
  
<body>  
  
<div id="map">  
 <script type="text/javascript">  
  
 //Importing the mapbox tiles layer. For our purposes, the example map is sufficient  
 **var** mapboxTiles = L.tileLayer('https://{s}.tiles.mapbox.com/v3/examples.map-zr0njcqy/{z}/{x}/{y}.png', {  
 attribution: '<a href="http://www.mapbox.com/about/maps/" target="\_blank">Terms &amp; Feedback</a>'  
 });  
  
 //Importing the map from leaflet and adding the tiles layer.  
 //The initial view is centered to Tel Aviv, according to the example info (lines 171 and 222).  
 **var** map = L.map('map')  
 .addLayer(mapboxTiles)  
 .setView([32.07452, 34.78875], 13);  
  
 // Initialize the SVG layer  
 map.\_initPathRoot()  
  
  
 // Categorical colors for 2 lines routes and the optional changing station's colors  
 // 3 color hues were chosen using ColorBrewer  
 **var** mappingColors =  
 {  
 "line1Color": "#e41a1c",  
 "line2Color": "#377eb8"  
 };  
  
 // Third color, to be sequential and used to dsitinguish between the best and worst change stations  
 **var** greens = ['rgb(247,252,245)', 'rgb(229,245,224)', 'rgb(199,233,192)', 'rgb(161,217,155)', 'rgb(116,196,118)', 'rgb(65,171,93)', 'rgb(35,139,69)', 'rgb(0,90,50)'];  
  
 //Holds the hues that will be assigned for the waiting time mapping (the station changeStations color).  
 //The actual setting is done after the relevant number of optional changing station is calculated.  
 **var** stationsHues;  
  
 // We pick up the SVG from the map object  
 **var** svg = d3.select(map.getPanes().overlayPane).append("svg"),  
 g = svg.append("g").attr("class", "leaflet-zoom-hide");  
  
 //Styling variables used for the routes path layer  
 **var** Style501 = {  
 "color": mappingColors.line2Color  
 };  
  
 **var** Style171 = {  
 "color": mappingColors.line1Color  
 };  
  
 //Drawing the routes' paths using leaflet layer and not svg  
 d3.json("stopShapes61551\_501.geojson", **function** (collection) {  
  
 L.geoJson(collection, {style: Style501}).addTo(map);  
 });  
  
 d3.json("stopShapes52473\_171.geojson", **function** (collection) {  
 L.geoJson(collection, {style: Style171}).addTo(map);  
 });  
  
 //Using SVG, iterating the stations and displaying only the  
 //change stations (in this example there are only 8 stations)  
 d3.json("stopTimesBothLines501\_171.geojson", **function** (geoShape) {  
  
 // create a d3.geo.path to convert GeoJSON to SVG  
 **var** transform = d3.geo.transform({point: projectPoint}),  
 path = d3.geo.path().projection(transform);  
  
  
 // create path elements for each of the features  
 stations = g.selectAll("path")  
 .data(geoShape.features)  
 .enter().append("path")  
 .data(geoShape.features)  
 .attr("class", "lineStations")  
  
 //Add circles for the change stations, filtered by having a waiting time ranking  
 changeStations = g.selectAll("circle")  
 .data(geoShape.features  
 .filter(**function** (d) {  
 **return** (d.properties.WAIT\_RANK != undefined)  
 }))  
 .enter()  
 .append("svg:circle")  
  
 //Adding text for each of the origin, destination and change stations  
 stationName = g.selectAll("text")  
 .data(geoShape.features  
 .filter(**function** (d) {  
 **return** (d.properties.WAIT\_RANK != undefined )  
 }))  
 .enter()  
 .append("svg:text")  
 .text(**function** (d) {  
 **return** d.properties.STOP\_NAME;  
 })  
 .attr("class", "stationsText")  
  
 //Calculating the minimum and maximum waiting ranks for the color scaling domain  
 minVal = d3.min(geoShape.features, **function** (d) {  
 **if** (d != undefined)  
 **return** d.properties.WAIT\_RANK;  
 **return** Number.MAX\_VALUE;  
 });  
  
 maxVal = d3.max(geoShape.features, **function** (d) {  
 **if** (d.properties.WAIT\_RANK != undefined)  
 **return** d.properties.WAIT\_RANK  
 **else  
 return** 0;  
 });  
  
 numOfChangeStations = geoShape.features.filter(**function** (d) {  
 **return** (d.properties.WAIT\_RANK != undefined)  
 }).length;  
  
 //Scaling the waiting rank to the generated color scale  
 stationsHues = d3.scale.quantize()  
 .domain([minVal, maxVal])  
 .range(greens);  
 //Alerting the user that only up to 9 change stations will be displayed  
 //only if there are more than 9 in the data  
 **if** (numOfChangeStations > 9)  
 alert("Showing top 9 stations with shortest change time");  
  
 //setting the map reset function for refreshing the view after zoom & pan  
 map.on("viewreset", reset);  
  
 reset();  
  
  
 **function** reset() {  
  
 // fit the SVG element to leaflet's map layer  
 bounds = path.bounds(geoShape);  
  
 **var** topLeft = bounds[0],  
 bottomRight = bounds[1];  
  
  
 svg.attr("width", bottomRight[0] - topLeft[0])  
 .attr("height", bottomRight[1] - topLeft[1])  
 .style("left", topLeft[0] + "px")  
 .style("top", topLeft[1] + "px");  
  
 g.attr("transform", "translate(" + -topLeft[0] + ","  
 + -topLeft[1] + ")");  
  
  
 //updating the station names and shapes locations according  
 stationName.attr("x", **function** (d) {  
 **return** path.centroid(d)[0];  
 })  
 .attr("y", **function** (d) {  
 **return** path.centroid(d)[1];  
 })  
  
 changeStations.attr("cx", **function** (d) {  
 **return** path.centroid(d)[0];  
 })  
 .attr("cy", **function** (d) {  
 **return** path.centroid(d)[1];  
 })  
 .attr("r", **function** (d) {  
 **return** ((maxVal - d.properties.WAIT\_RANK)) \* 2;  
 })  
 .style("fill", **function** (d) {  
 **return** stationsHues(maxVal + 1 - d.properties.WAIT\_RANK);  
 })  
 .attr("class", "stations")  
 .append("svg:title")  
 .text(**function** (d) {  
 **return** "Waiting time: " + (d.properties.WAIT\_TIME);  
 });  
 }  
  
  
 // Use Leaflet to implement a D3 geometric transformation.  
 **function** projectPoint(x, y) {  
 **var** point = map.latLngToLayerPoint(**new** L.LatLng(y, x));  
 **this**.stream.point(point.x, point.y);  
 }  
 })  
  
  
 </script>  
</div>  
</body>  
  
</html>

### Results



### Advantages, Disadvantages & Alternatives

The output clearly shows that the biggest and darkest circle is that of Hamasger station (bottom station). Therefore, this station is the most recommended for the user based upon the “best change station” definition.

However, there are several (potential) disadvantages pointed by myself and classmates who viewed this visualization:

* The double mapping of the station’s ranking might confuse users as to why there are variations in both size and color.
* The color/size range might not be trivial. A legend is needed to state clearly the range edges.
* Due to the map’s base layers colors, the red and blue colors are not distinctive enough.
* It is not clear which path represents which route and where is the origin and destination

All these issues were addressed in iteration 2.

## Iteration 2

### Data to Visual Attributes Mapping

Changes/additions to iteration 1 mapping:

Origin & Destination stations -> shape: ellipses

Change station ranking -> sequential circle colors **only** (size is constant for all circles)

Ranking domain -> sequential color legend

Lines coloring was changed to other categorical colors which are more distinct over a map:

#984ea3 and #ff7f00

Tooltips were added to display the waiting time and the route numbers

### Code

This section covers only the additions and changes to the first iteration code. The order might appear differently in the end source code

Adding the legend:

/\*Legend styling\*/  
.legend {  
 padding: 6px 8px;  
 font: 14px/16px Arial, Helvetica, sans-serif;  
 background: white;  
 background: rgba(255, 255, 255, 0.8);  
 box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);  
 border-radius: 5px;  
 line-height: 18px;  
 color: #555;  
}

//Adding a legend with sequential horizontal bar for easy access  
**var** legend = L.control({position: 'bottomright'});  
  
legend.onAdd = **function** (map) {  
  
 **var** div = L.DomUtil.create('div', 'legend');  
 div.innerHTML += '<p style="margin:auto; background: linear-gradient(to right, ' + greens + ')"</p>' + '<br>';  
 div.innerHTML += '<p style="font-weight: bold"> longest waiting &#8596 shortest waiting</p>';  
 **return** div;  
};  
  
legend.addTo(map);

Changing the routes colors:

// Categorical colors for 2 lines routes and the optional changing station's colors  
// 3 color hues were chosen using ColorBrewer  
**var** mappingColors =  
{  
 "line1Color": "rgb(152,78,163)",  
 "line2Color": "rgb(255,127,0)"  
};

Adding the origin & destination shape and text:

// Add ellipses for indicating the origin and destination stations specified by the user  
origDestStations = g.selectAll("ellipse")  
 .data(geoShape.features  
 .filter(**function** (d) {  
 **return** (d.properties.IS\_ORIGIN == "TRUE" || d.properties.IS\_DEST == "TRUE")  
 }))  
 .enter()  
 .append("svg:ellipse")  
  
//Adding text for each of the origin, destination and change stations  
stationName = g.selectAll("text")  
 .data(geoShape.features  
 .filter(**function** (d) {  
 **return** (d.properties.WAIT\_RANK != undefined ||  
 d.properties.IS\_ORIGIN == "TRUE" || d.properties.IS\_DEST == "TRUE")  
 }))  
 .enter()  
 .append("svg:text")  
 .attr("dx", "12")  
 .attr("dy", "4")  
 .text(**function** (d) {  
 **var** stopName = d.properties.STOP\_NAME;  
 **if** (d.properties.IS\_ORIGIN == "TRUE")  
 stopName = "מוצא: " + stopName;  
 **if** (d.properties.IS\_DEST == "TRUE")  
 stopName = "יעד: " + stopName;  
 **return** stopName;  
 })  
 .attr("class", "stationsText")

origDestStations.attr("cx", **function** (d) {  
 **return** path.centroid(d)[0];  
})  
 .attr("cy", **function** (d) {  
 **return** path.centroid(d)[1];  
 })  
 .attr("rx", 10)  
 .attr("ry", 7)  
 .attr("class", "stations")  
 .style("fill", "#ffff33")  
 .append("svg:title");

Adding routes numbers tooltips:

// initialize & update the path data  
stations.attr("d", path)  
 .attr("class", "lineStations")  
 .text("Line Number");

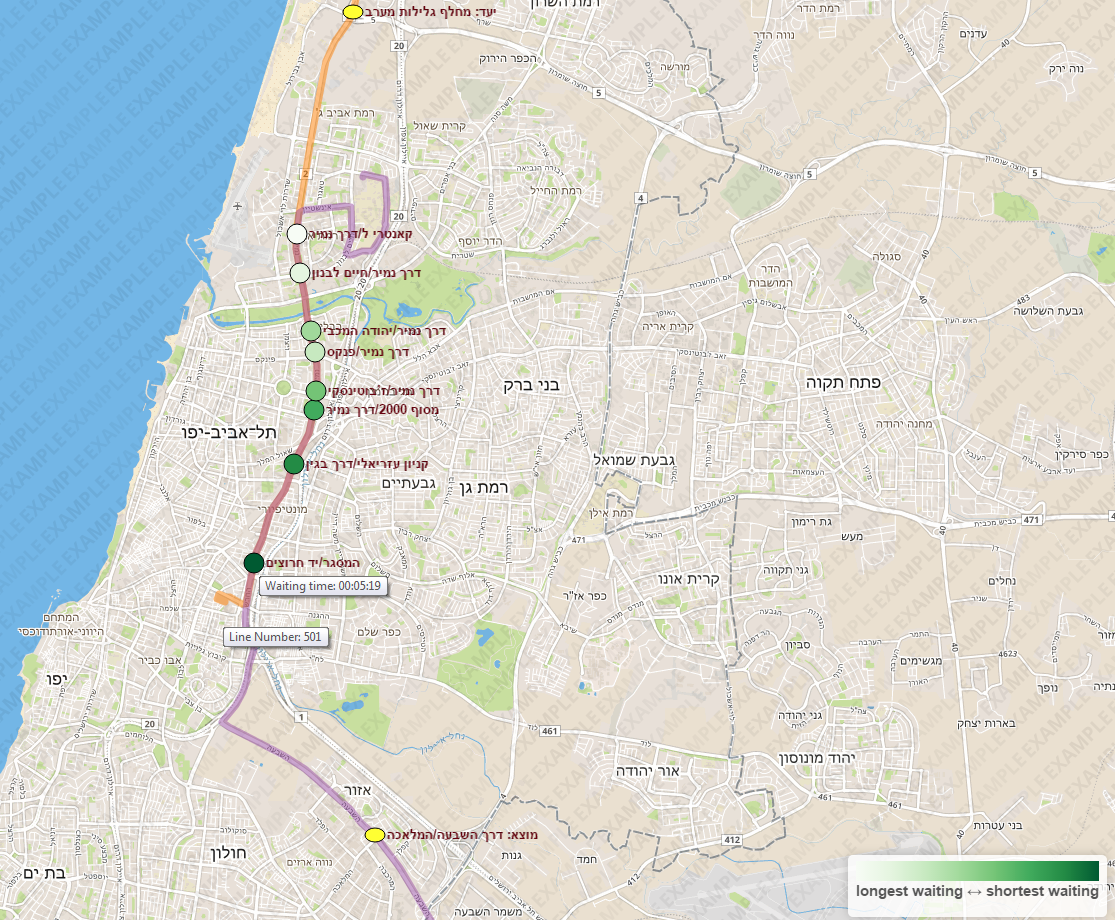
//adding line numbers in tooltip of the lines  
stations.append("svg:title")  
 .text(**function** (d) {  
 **return** "Line Number: " + (d.properties.LINE\_NUM);  
 });

Setting the change stations circle radius to constant

changeStations.attr("cx", **function** (d) {  
 **return** path.centroid(d)[0];  
})  
 .attr("cy", **function** (d) {  
 **return** path.centroid(d)[1];  
 })  
 .attr("r", 10)

Final Code can be viewed in the [Git Repository in file Index.html](https://github.com/shakedk/shakedk.github.io)

### Results



### Advantages, Disadvantages & Alternatives

The map now looks less cluttered with differently sized circles while finding the best station is easier thanks to the legend. The addition of the origin and destination stations adds a point of reference for the user. Tooltips supply optional information for users who are interested in concrete data in addition to the ranking visualization.

# Evaluation

## Reflection on Design Decisions

Following are several reflections I had during the data visualization development:

* **Station shapes**

I chose circles to represent the station for several reasons. Firstly, a basic geometric shape provides simplicity and the ability to fill its area with color for sequential scaling. Moreover, circles are common markers in cartography and therefore pose an easy-to-work-with shape for users.

* **Limiting number of stations**

Many bus routes crisscross and overlap each other, which might supply our data visualization with many change station to rank and visualize. I decided to map the stations ranking to one hue sequential scale for obvious reasons (divergent or categorical mapping do not apply for this type of data). Colorbrewer.org doesn’t offer a scale bigger than 9 colors for one-hue range. Therefore, I decided that only the top 9 change stations should be displayed. Apart from allowing the user to differentiate between colors easily, he doesn’t have to bother himself with a too much data that doesn’t give any advantage.

* **Double mapping**

At first, I believed that double mapping the ranking to both color and size will ease on the user’s decision making process and clearly state which station is the best. However, fellow classmates indicated that for them, first-time viewers, the double mapping is confusing. I then decided to keep the circles’ size constant and map the ranking to sequential color scale only.

* **Route numbers visualization**

The visualization is only a part of a larger system that should provide the user with the ability to choose origin and destination, offer different routes and display relevant information about the routes, trips and stations. However, as this is a demo system, I decided to add tooltips indicating which route is displayed for users to be able to understand the trip’s context.

* **Rank-color scaling direction – Open issue**

One open issue is which direction should the ranking be mapped to the color range: should the darkest color indicate the longest waiting time or the shortest waiting time (=the best change station)? I discussed with different people and got different responses.

This is probably a product of the fact that a darker color intuitively means “more data”. So, the question resides with whether the user interprets “more data” as “more waiting time” or “better option to choose”.

I believe the map’s legend solves this issue partially, but a full user study would provide the best choice for the default view.

## The Value of the Visualization

##### V = Time + Insights + Essence + Confidence

##### Time

This data visualization certainly saves the user time in comparison with classic scheduling tables and maps. Instead of comparing and computing the minimal waiting time, the information is displayed clearly and can be obtained quickly.

##### Insights

Users might not even consider the option that there are better and worse change stations, assuming that busses advance on the same pace more or less compared one to another for a given time of the day. The map helps to demonstrate the time differences (even on a straight road with neighboring stations).

Over time, users can find patterns of best areas to change stations at. With rather not-too-big modifications, the data be layered or aggregated so users can gain this information.

##### Essence

Long scheduling tables with textual routes and change times are minimized to several colored circles on colored paths. Users don’t need to match overlapping change stations or let automatic tools to choose the best change station, while not being compelled to go through a lot of information records.

##### Confidence

The map is layered with basic shapes and coloring to allow the user to gain insight about the ranking scale over line routes. I believe this is a simple and straight-forward approach that transmits to the user that the original data is displayed almost as-is, without major modifications. The main advantage of the visualization, in my opinion, is that the system doesn’t “decide” for the user which station is the best, but rather shows the best possibilities visually ranked for easy navigation and data extraction.

Code can be found on Git here: <https://github.com/shakedk/shakedk.github.io>

Demo can be found on: <http://shakedk.github.io/>