A Metric to Improve Seattle Metro Transit Bus Routes

Romain Li, Richard McGovern, Andrew Walker

August 23, 2013

Abstract

In this paper we develop a metric to be used to assist in the process of reducing or cutting routes to a transit system in the event of a budget cut. The metric measures efficiency, level of service, and fairness, then creates a system where one can make reductions while having a simple tool for determining how "good" a given route is. Finally we apply this metric to the Seattle Metro system, several million data points, under a few different cases in order to illustrate the potential uses.

Contents

1 Introduction

Due to massive decreases in funding following the recession, Metro Bus Transit has been forced to implement a potential series of reduction in bus service. They estimate up to 17% of the current Metro system could be eliminated in order to balance their budget [3]. Metro has decided to make reductions but still provide efficient service that achieves three goals in their new policy framework:

1. Providing productive service that carries more people per hour (efficiency)

2. Serving communities that depend heavily on transit (service)

3. Distributing service fairly though out the county (fairness) [4]

In this paper we will be developing and demonstrating a metric that allows us to suggest changes (either reductions or cuts) to the current system and the impact they have in these three areas. When applying the metric we suggest using the current system as a baseline. We assume that doing as well as the system does right now is acceptable. This is a reasonable assumption because holistically the system works. This means that the efficiency, fair distribution, and service of dependent communities are assumed to be at a fair level. Without this assumption we would have to work to create what would be an ideal or optimal balance of these categories to rank against, but this is a huge task which we are not even sure has a good answer.

2 Developing a Metric

2.1 Elements of the Metric

This metric should be able to measure the current state of the Metro system. Specifically it should combine elements of each of the three categories into a single statistic. The goal of the metric would be to give a user an idea of which routes are current being scheduled or run inefficiently. For example if there were to be two identical routes in a given system, running the metric would give a result showing that those routes were inefficient. The same result should be expected if we tweak the system such that any of the given categories take a hit, for example if there was a route with an extremely low on-time percentage, then said route would score very poorly. The metric should also be flexible enough to allow a user to tailor the weights and place emphasis on different aspects of a system if they would like to. As a result our metric is a linear combination of a measurement of efficiency and a measurement of service, with fairness being used a check to make sure any cuts made are fair. This is computed for each route over a given system.

2.1.1 Efficiency

When we talk about efficiency we mean how efficient a given bus is in terms of how similar a route is to others (for example if two routes share a large number of stops), on-time percentage, number of passengers, and cost. We will begin thinking about a simple model for computing a measure of efficiency.

2.1.2 Naive Model

We can simply create a ratio of the positive statistics over the negative statistics. This is a good baseline to start with as it gives an intuitive idea of efficiency. In our case that means if we let T_n be the proportion of time that a route n is on-time, P_n be the the number of people carried by route n per hour, R_n be the redundancy factor (maximum number of shared stops over all other routes) of route n, and C_n be the hourly cost of route n, then our naive efficiency metric, $E(n)$ would take a general form of:

$$E(n) = \alpha P_n + \beta T_n - \gamma C_n - \delta R_n \tag{1}$$

for some scaling constants α, β, γ, and δ to be determined empirically or used to weight importance of each aspect. For example if you value on-time percentage more, have β be a larger constant relative to the rest.

2.1.3 A Possible Improvement on the Naive Model

One improvement to this would be to choose a better redundancy factor. Currently our redundancy factor simply tells us the maximum number of shared stops a route n has with any other route. This could be a poor choice because it does not take into account that there could exist a route which shares a lot of stops, but then travels to a remote area where no other routes reach. The current factor would rate this route poorly, when in reality it is quite good at reaching non-redundant stops. One such improvement we would consider would be to not only look at shared stops, but also factor in the unique stops. This would balance out any route which fell into the above category, saving the rating if it did indeed reach unique stops.

Another improvement that could be made would be to take all routes into account simultaneously. Currently we maximize over all routes, but take no special care to identify how many routes pass through any given stop. If we weighted the calculation to care more about a stop as more buses travel through it, then we could have a factor which scaled with both the size of the system and accounts for not only unique stops, but semi-unique, that is any stop which only has a few buses would provide a larger rating boost than one with many common buses.

2.1.4 Serving Dependent Communities

In order to measure how much a given route, n, serves a community we chose to look at the set of stops a given route serves and determine which district of the city each stop is located in. In each district, d, x_d people get on a bus and y_d people get off of a bus everyday. Let D_n be the set of districts route n stops in, then our service score, $S(n)$ will be defined as follows:

$$S(n) = \sum_{d \in D_n} x_d + y_d \tag{2}$$

We chose this to be the score we assign because it takes into account both the people who go to a certain district and those who come from a certain district, both of which are equally important when determining if a community is served by a given route. Using both the ons and offs as opposed to just one also allows us to more accurately assess who uses the system. If we tried to characterize this with just one, we would be missing a lot of information. At first glance it is easy to assume that the number of people getting on should be roughly equal to the number of people getting off, which would lead one to think that our score is more inflated than it needs to be. This seems reasonable, if someone using a bus to go somewhere, then they likely use it to return. However, we are more accurately able to account for people who use multiple modes of transportation, for example, people who carpool to or from work, but use the Metro system to go the other direction.

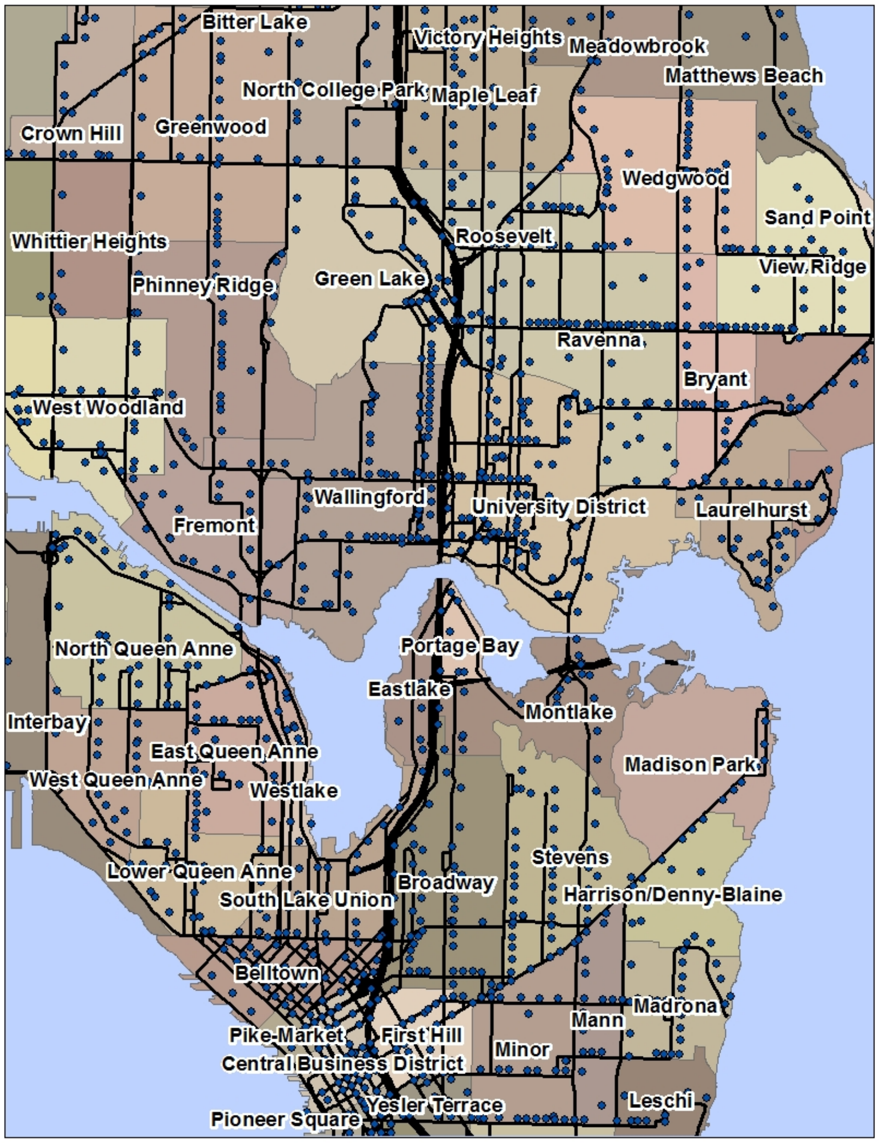

Figure 1: North Seattle split into districts with bus routes and bus stops marked.

2.1.5 Fair Distribution

Unlike the other two areas, fairness is a characteristic of a system as a whole. The way we will use this aspect of Metro's goals is to utilize it as a check to ensure that the resulting system is still fair. It is difficult to use this as a measure before any change has happened, so instead we apply it afterword.

We begin by assuming that the current system is "fair enough" and that any changes to the system will create an imbalance, unless balanced out by other changes to different areas of service. For example if we were to suggest cutting ten routes from one district, this would not be considered fair. In order to balance it out we would have to suggest cutting routes from other districts to disperse the amount of cuts uniformly though out the system. We accomplish this by looking at how many routes run through each district and strive to keep the ratio of routes through the district to the total number of routes roughly equal to what it is when we begin the process. If we deviate too much, then it is highly likely that we hit some areas too hard and others not hard enough.

One concern with this might be that this type of structure incentivizes cutting more routes in order to maintain a high fairness rating. However we are constrained by budget. We only want to cut until we are within our projected budget guidelines. This removes the somewhat perverse incentive of cutting more routes to improve our rating and creates a system that promotes even cuts and reductions, even if a particular area has extremely inefficient routes, because inefficient routes are better than no routes in terms of service.

2.2 Implementation

When combining these three areas into one metric we first address the fact that they are in completely different scales. In order to normalize the scores we compute the individual statistics and then convert them into z-scores. A z-score is the number of standard deviations away from the mean a particular piece of data is. We did this because any measurement is only really going to mean anything if it is relative to its own data set. With a z-score we can see how a route performs relative to the others within the same system, there is no standard for a transportation system and creating one is a hopeless task, so self-relativity is the best we can do. Z-scores also are normalized, which means that when we directly compare our statistics they are in the same units, standard deviations from the mean, as opposed to being measured in anything from people carried per revenue hour to simple counting numbers.

Combining the measures of efficiency and service, equations (1) and (2) respectively, we can produce a linear combination whose coefficients can be chosen with the particular problem, system, and situation in mind:

$$\zeta\left(\alpha P_n + \beta T_n - \gamma C_n - \delta R_n\right) + \epsilon \sum_{d \in D_n} x_d + y_d \tag{3}$$

With this measure for each route one can see which routes are the least effective at achieving Metro's overall goals, and consider cutting or reducing those routes. Then after the proposed changes are in mind, using the third area, fairness, we can check whether or not the changes are fair. The whole process would look like this:

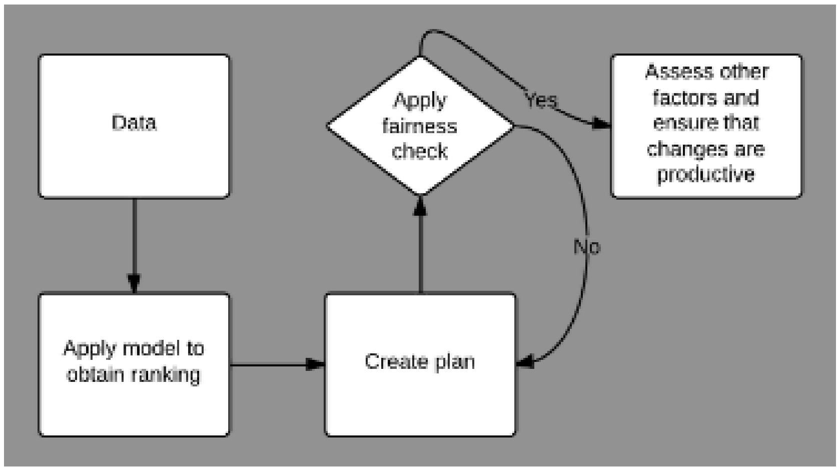
Data
Apply fairness check
Yes
Assess other factors and ensure that changes are productive
No
Apply model to obtain ranking
Create plan

Figure 2: Flowchart of the process to utilize our metric in planning.

3 Applying the Metric

If we set all of the coefficients to be equal to one and use Seattle Metro Transit's spring quarter data with our model we get measurements for each of Metro's current routes.

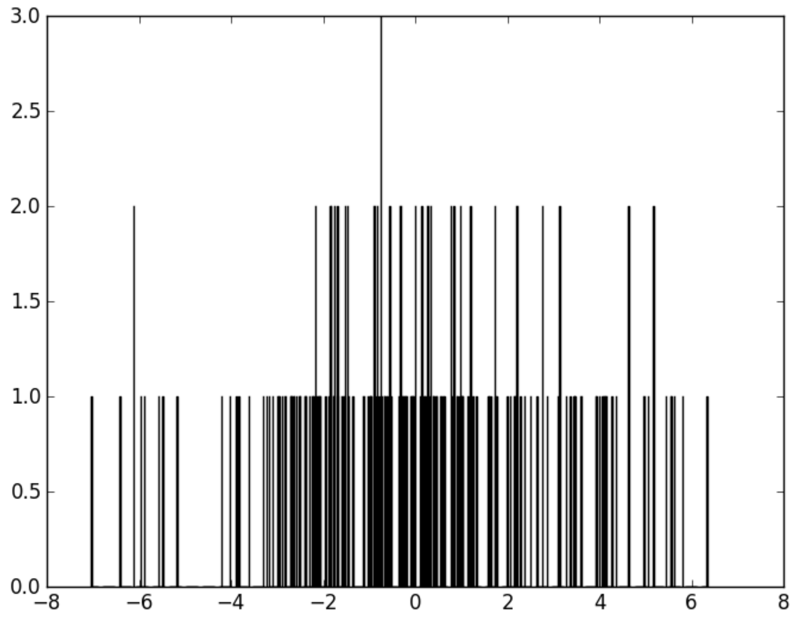

Figure 3: Distribution of scores when $\alpha = \beta = \gamma = \delta = \epsilon = \zeta = 1$

The ratings rank from -7.0395962076 to 6.34163756884 with a mean of $9.70005968552e-15$, median of -0.188773525527, and standard deviation of 2.62917998767. This set up means that we are weighing each element of efficiency equally and efficiency as a whole equal to service as a whole. In the figure we can see that there is a distinct group of low scoring routes.

If we use $\alpha = 3$, $\beta = 1$, $\gamma = 1$, $\delta = 2$, $\epsilon = 11$, and $\zeta = 13$ we get a distribution like this:

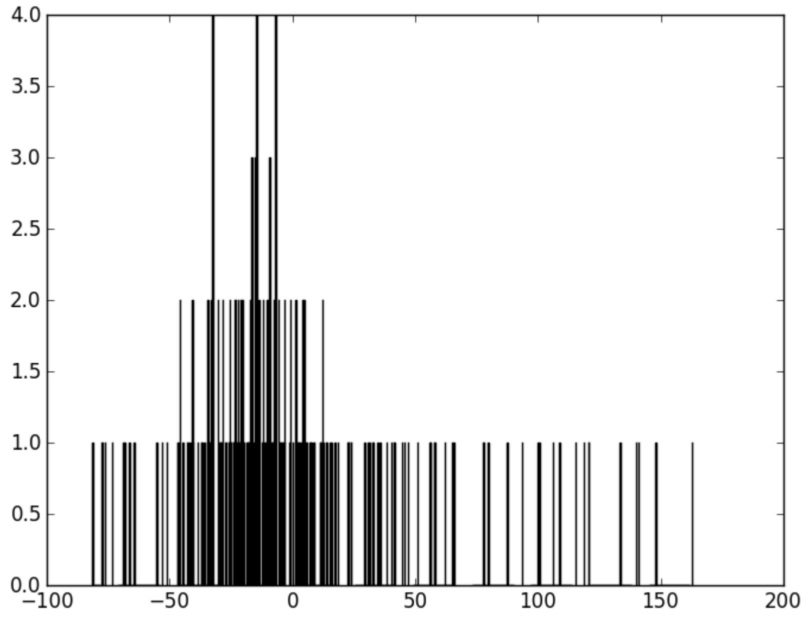

Figure 4: The mean score is 6.89160662693e-15, the median score is -9.72340545236, the STD is 44.1898912686, the maximum score is 162.957583613, the minimum score is -81.4413187051

This distribution weighs efficiency slightly heavier than service, and within efficiency it weighs passengers carried per revenue hour the highest, then redundancy, and finally cost and on-time percentage the least. There is still a group which lies distinctly below the rest, most of these are the same routes as in the previous run. If these continue to score low in other configurations then they are likely very good candidates for cuts, reductions, or revisions to bring them up to par with the rest of the routes in the system.

4 Improvement to Redundancy

Define a route vector $\mathbf{v}^{(k)}$ given a bus-stop graph such that the i^{th} entry is 1 if the route goes through this bus-stop, otherwise it is 0, where k is the route number and n is the number of bus-stops from the graph.

Define a bus-stop matrix $A_{n\times n}$ to be the matrix given a bus-stop graph such that the i, j entry is 1 if there is a direct route from i to j and 0 otherwise, where i and j are the bus-stops from the graph and n is the number of bus-stops.

To compare two routes our measure M is:

$$M = ||A\mathbf{v}^{k_1} - A\mathbf{v}^{k_2}|| \tag{4}$$

to turn this into a metric for the system we simply partition the routes and average this measure on the elements of the set, comparing your route n to every other route.

$$R_n = \frac{\sum_{i \neq n} |A\mathbf{v}^{(n)} - A\mathbf{v}^{(i)}|}{n} \tag{5}$$

This gives a more comprehensive model which considers uniqueness as well as redundancy.

5 Conclusion

We have presented a metric for ranking bus routes in an arbitrary system in a way that is highly flexible and able to applied to nearly any transit system. Working with coefficients one can customize the metric to emphasize the particular needs of the system. While we do not claim to be able to efficiently or effectively select which routes to cut or reduce, this is a helpful tool for policy makers and Metro administrations to use to get a feel for which routes are in need of changes to keep up with others in the system.

6 References

[1] Ceder, Avishai and Nigel H. M. Wilson, *Bus Network Design* Pergamon Journals Ltd., 1986.

[2] Karimi, Hassan A., Ratchata Peachavanish, and Jun Peng, *Finding Optimal Bus Service Routes: Internet-Based Methodology to Serve Transit Patrons* DOI: 10.1061/(ASCE)0887-3801, 2004

[3] King County Metro, *Moving Toward Financial Stability and Sustainability.* 2011. url: http://metro.kingcounty.gov/am/future/service-cuts.html. Date Accessed: July 7, 2013.

[4] King County Metro, *King County Metro Strategic Plan.* 2013. url: http://metro.kingcounty.gov/planning/pdf/KCMT_ServiceGuidelines_07-11-11.pdf. Date Accessed: 7/25/13

[5] Per-Ake Andersson and Gian-Paolo Scalia-Tomba, *A Mathematical Model of an Urban Bus Route*, Pergamon Press Ltd., 1981.

7 Appendix

7.1 example.py

```
import csv
import numpy as np
import scipy as sp
from scipy import stats
import matplotlib.pyplot as plt

#################### Initialize model variables. ####################
alphaVal = raw_input('Enter an alpha value: ')
betaVal = raw_input('Enter a beta value: ')
gammaVal = raw_input('Enter a gamma value: ')
deltaVal = raw_input('Enter a delta value: ')
epsVal = raw_input('Enter an epsilon value: ')
zetaVal = raw_input('Enter a zeta value: ')

# A map from route numbers (str) to a set of stops (str).
routeToStop = {}
with open('routeStops.csv', 'rU') as data:
    reader = csv.reader(data)
    for row in reader:
        if not row[0] in routeToStop.keys():
            routeToStop[row[0]] = set(row[1].upper())
        else:
            routeToStop[row[0]].add(row[1].upper())
    for sets in routeToStop.values():
        temp = []
        for items in sets:
            if len(items) < 5:
                temp.append(items)
        for items in temp:
            sets.remove(items)

# A set of all stops (str).
stops = set()
for sets in routeToStop.values():
    for intersection in sets:
        stops.add(intersection)
stops = sorted(stops)

# A set of all routes (str).
tempRoutes = set()
for rt in routeToStop.keys():
    # Filler for code folding.
    tempRoutes.add(int(rt))
tempRoutes = sorted(tempRoutes)
routes = []
for rt in tempRoutes:
    # Filler for code folding.
    routes.append(str(rt))


# A map from stops (str) to district (str).
stopToDistrict = {}
with open('OnCrossDistrict.csv', 'rU') as data:
    reader = csv.reader(data)
    for row in reader:
        stopToDistrict[row[0].upper() + ' & ' + row[1].upper()] = row[2]

# A set of all districts (str).
districts = set()
for dist in stopToDistrict.values():
    # Filler for code folding.
    districts.add(dist)

# A map from district (str) to a set of route (str).
districtToRoute = {}
for dist in districts:
    routeSet = set()
    for rt in routes:
        for st in routeToStop[rt]:
            try:
                if stopToDistrict[st] == dist:
                    routeSet.add(rt)
            except KeyError:
                pass
    districtToRoute[dist] = routeSet

# A map from district (str) to number of passengers, ons and offs,
# per hour (float).
districtToUsage = {}
with open('OnsOffsPerRoute.csv', 'rU') as data:
    reader = csv.reader(data)
    temp = next(reader)
    for row in reader:
        try:
            if not stopToDistrict[row[8] + ' & ' + row[10]] \
            in districtToUsage.keys():
                districtToUsage[stopToDistrict[row[8] + ' & ' + row[10]]] \
                = float(row[5]) + float(row[6])
            else:
                districtToUsage[stopToDistrict[row[8] + ' & ' + row[10]]] \
                += float(row[5]) + float(row[6])
        except KeyError:
            pass

# A map from route (str) to cost per hour (float).
routeToCost = {}
with open('TimeDistanceLoadofRoute.csv', 'rU') as data:
    reader = csv.reader(data)
    temp = next(reader)
    temp = next(reader)
    for row in reader:
        if not row[0] in routeToCost.keys():
            try:
                routeToCost[row[0]] = float(row[18])
            except ValueError:
                pass

# A map from route (str) to load per hour (float).
routeToLoad = {}
with open('TimeDistanceLoadofRoute.csv', 'rU') as data:
    reader = csv.reader(data)
    temp = next(reader)
    temp = next(reader)
    for row in reader:
        if not row[0] in routeToLoad.keys():
            routeToLoad[row[0]] = float(row[11])
        else:
            routeToLoad[row[0]] += float(row[11])

# A map from route (str) to redundancy (int).
routeToRedundancy = {}
for rt in routes:
    temp = set()
    for stuff in routes:
        temp.add(stuff)
    temp.remove(rt)
    maxChain = 0
    for rt2 in temp:
        chain = set(routeToStop[rt]).intersection(set(routeToStop[rt2]))
        if len(chain) > maxChain:
            maxChain = len(chain)
    routeToRedundancy[rt] = maxChain

# A map from route (str) to on time percentage (float).
routeToOTP = {}
with open('OTPByRoute.csv', 'rU') as data:
    reader = csv.reader(data)
    count = 1
    while count < 4:
        temp = next(reader)
        count += 1
    while count < 221:
        temp = next(reader)
        if temp[0][:-1] in routeToOTP.keys():
            routeToOTP[temp[0][:-1]] = \
            np.average([routeToOTP[temp[0][:-1]], float(temp[6][:-1])])
        else:
            routeToOTP[temp[0][:-1]] = float(temp[6][:-1])
        count += 1

# A map from route (str) to a set of districts (str) the route
# passes through.
routeToDistrict = {}
for rt in routes:
    dist = set()
    temp = routeToStop[rt]
    for st in temp:
        try:
            dist.add(stopToDistrict[st])
        except KeyError:
            pass
    routeToDistrict[rt] = dist

# A map from route (str) to a sum of usage across all districts
# the route passes through (float).
routeToService = {}
for rt in routes:
    rtSum = 0
    for dist in routeToDistrict[rt]:
        rtSum += districtToUsage[dist]
    routeToService[rt] = rtSum

########################################################################

# The passenger load for each route.
P = []
for rt in routes:
    P.append(routeToLoad[rt])
zP = stats.zscore(P)

# These were left out of metro's data, we put in an average number.
leftOut = ['522', '540', '542', '545', '550', '554', '555', '556',
           '560', '671', '672', '673', '674']
for s in leftOut:
    routeToOTP[s] = 77.0

# The on time percentage for each route.
T = []
for rt in routes:
    T.append(routeToOTP[rt])
zT = stats.zscore(T)

# The hourly cose for each route.
C = []
for rt in routes:
    C.append(routeToCost[rt])
zC = stats.zscore(C)

# The redundancy factor of each route.
```

```python
R = []
for rt in routes:
    R.append(routeToRedundancy[rt])
zR = stats.zscore(R)

# The service score of each route.
S = []
for rt in routes:
    S.append(routeToService[rt])
zS = stats.zscore(S)

# Processing and displaying data.
measures = []

for k in range(0,len(routes)):
    measure = float(zetaVal)*((float(alphaVal)*zP[k] + \
              float(betaVal)*zT[k]) + (float(gammaVal)*zC[k] + \
              float(deltaVal)*zR[k])) + \
              float(epsVal)*zS[k]
    measures.append(measure)
    print 'Route ' + routes[k] + ' has a measure of ' + str(measure)

print 'The mean score is ' + str(np.mean(measures))
print 'The median score is ' + str(np.median(measures))
print 'The STD is ' + str(np.std(measures))
print 'The maximum score is ' + str(max(measures))
print 'The minimum score is ' + str(min(measures))

hist, bins = np.histogram(measures, bins = 1000)
width = 0.7 *(bins[1]-bins[0])
center = (bins[:-1]+bins[1:])/2
plt.bar(center, hist, align = 'center', width = width)

plt.show()
```