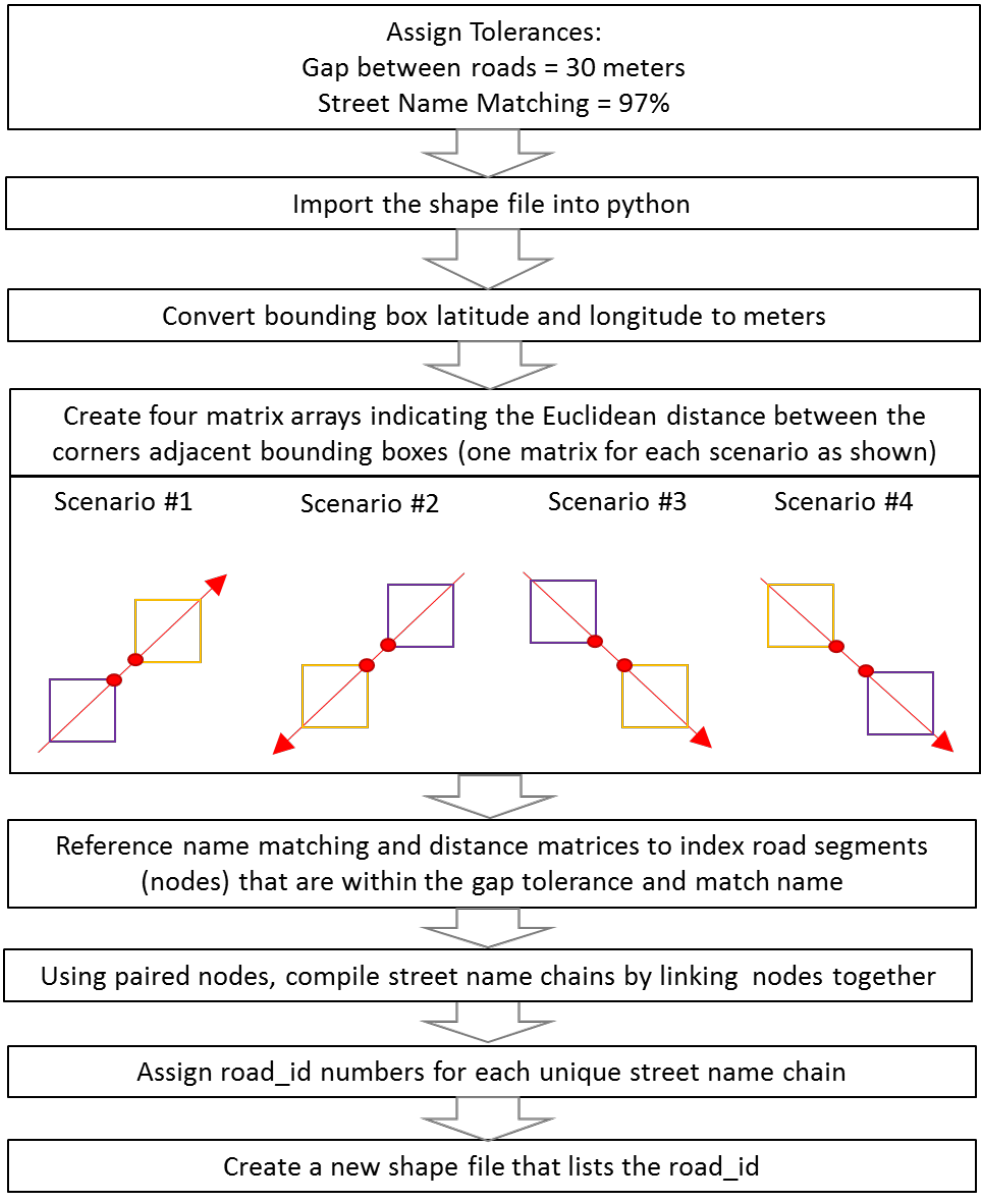
Harrison Durbin

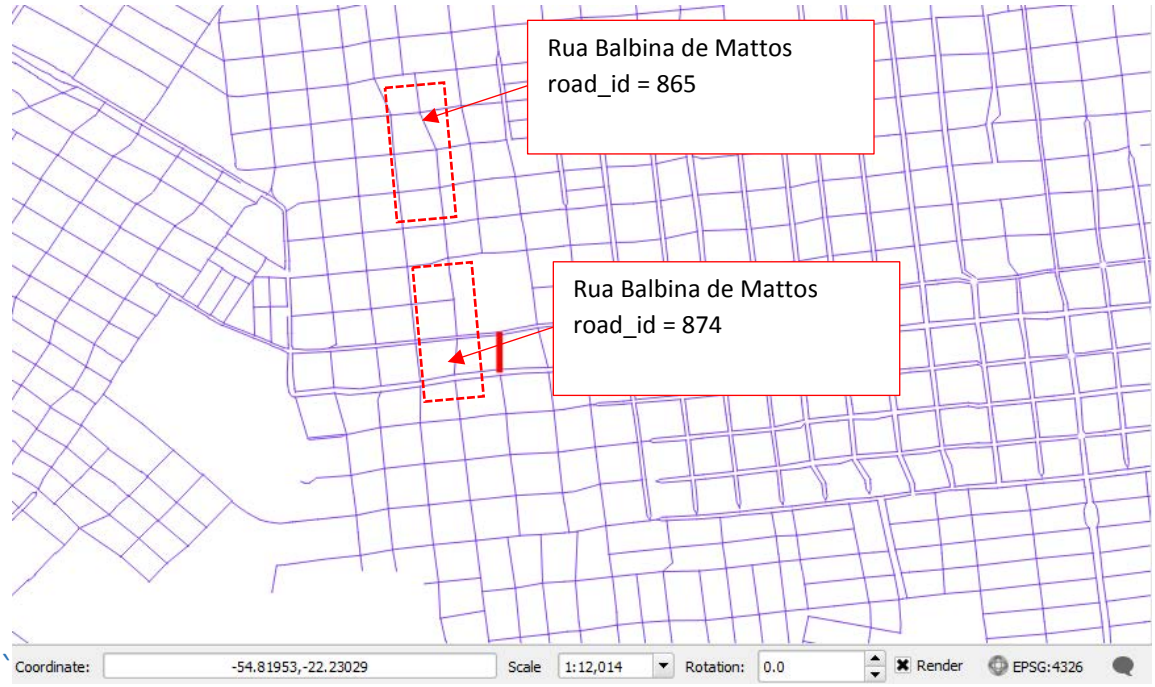

# TomTom Technical Test

As part of the TomTom Technical Test for the GIS Engineer position, an automated procedure to generate a unique "road_id" per "street name chain" was created using python only and checked using QGIS. The code was written using ipython notebook (code attached). The following shows the procedure used:




Four separate distance array were created because line segments may be at various angles. In each scenario, for each node (line segment), the edges (adjoining line segments) were indexed if meeting the tolerance criteria of gap and street name. A gap of 30 meters was assumed and 97% name matching was required. The data was then compiled to form street name chains and each chain was assigned a unique id.

# Appendix A – GIS Map Sample



# Appendix B – Python Code

# Harrison_Durbin_GISEng_Practical_Test_REVISED

November 6, 2016

## 1 Harrison Durbin Technical Test - TomTom - 05.11.2016

In [1]:
```
import shapefile
import numpy as np
from scipy.spatial.distance import cdist
import time
import difflib as dl
import sys
import winsound
```

In [2]:
```
# assign a value for the gap tolerance between road segments
gap_tolerance = 30 # meters

# assign a value for how closely the names of streets must match in street name chains
name_tolerance = 97 # percent matching
```

In [3]:
```
# import the shapefile
def load_shape(sf1):
    sf = shapefile.Reader(sf1)
    shapes = sf.shapes()
    records = sf.records()

    # create an array of the bounding box latitude and longitudes for each road segment
    bbox = np.zeros(shape=(len(shapes),4))
    for i in range(len(shapes)):
        bbox[i] = shapes[i].bbox

    # convert bounding box latitude and longitude coordinates into meters to calculate gaps bet
    coords = np.zeros(shape=(len(records),4)) # coords in meters of bounding box

    for i in range(len(records)):
        coords[i,0] = int((-bbox[i][0] - 54)*103262) # longitude converted to meters
        coords[i,1] = int((-bbox[i][1] - 22)*110730) # latitude converted to meters
        coords[i,2] = int((-bbox[i][2] - 54)*103262) # longitude converted to meters
        coords[i,3] = int((-bbox[i][3] - 22)*110730) # latitude converted to meters


    return records, shapes, coords

# function to load csv files into python
def iter_loadtxt(filename, delimiter=',', skiprows=0, dtype=float):
    def iter_func():
```

```
        with open(filename, 'r') as infile:
            for _ in range(skiprows):
                next(infile)
            for line in infile:
                line = line.rstrip().split(delimiter)
                for item in line:
                    yield dtype(item)

        iter_loadtxt.rowlength = len(line)

    data = np.fromiter(iter_func(), dtype=dtype)
    data = data.reshape((-1, iter_loadtxt.rowlength))
    return data

# # this code only needs to be ran once, so it is commented out
# # create a matrix to hold the percentage matching of names for all street segments
def create_name_matrix(records):

    name_match_matrix = np.zeros(shape=(len(records),len(records)))

    for i in range(len(records)):
        for j in range(len(records)):
            s = int(100*((dl.SequenceMatcher(None, records[i][0], records[j][0])).quick_ratio()
            name_match_matrix[i,j]=s

    # save the name matching matrix as a csv file to load later
    np.savetxt("name_matrix.csv", name_match_matrix, fmt='%.3i',delimiter=",")
    print 'Finished created name matching matrix.'

    return name_match_matrix

# # filling in distance matrix (distances from top right to lower left of different bounding bo
# values are only filled in if the name matches and it is not the same road segment
# this only needs to be done once
def create_dist_matrix(records,name_match_matrix,coords,n):

    # # initialize a distance matrix to hold values of distances between street segments with a
    dist_matrix = np.zeros(shape=(len(shapes),len(shapes)))

    # initially set a large value for all distances
    dist_matrix[:,:] = 1e8

    for i in range(len(records)):
        for j in range(len(records)):
            s = name_match_matrix[i,j] # taking the name match percentage from matrix
            if s > name_tolerance and i!=j:

                if n == 1:
                    dist_matrix[i,j]=abs(int(cdist([coords[i][[0,1]]],[coords[j][[2,3]]])[0]))
                    if i==len(records) and j==len(records):
                        np.savetxt("dist_matrix1.csv", dist_matrix, fmt='%.3i', delimiter=",")

                if n == 2:
                    dist_matrix[i,j]=abs(int(cdist([coords[i][[0,3]]],[coords[j][[2,1]]])[0]))
```

```
                    if i==len(records) and j==len(records):
                        np.savetxt("dist_matrix2.csv", dist_matrix, fmt='%.3i', delimiter=",")

                if n == 3:
                    dist_matrix[i,j]=abs(int(cdist([coords[i][[2,3]]],[coords[j][[0,1]]])[0]))
                    if i==len(records) and j==len(records):
                        np.savetxt("dist_matrix3.csv", dist_matrix, fmt='%.3i', delimiter=",")

                if n == 4:
                    dist_matrix[i,j]=abs(int(cdist([coords[i][[2,1]]],[coords[j][[0,3]]])[0]))
                    if i==len(records) and j==len(records):
                        np.savetxt("dist_matrix4.csv", dist_matrix, fmt='%.3i', delimiter=",")

    print 'Finished created distance  matrix.'

    return dist_matrix


# this code compiles lists of the nodes adjacent to each side
def link_nodes(records,name_match_matrix,fn):

    connectionsA = [] # nodes connected to the left of the node
    connectionsB = [] # nodes connected to the left of the node

    dist_matrix = iter_loadtxt(fn)

    for j in range(len(records)):

        s = name_match_matrix[dist_matrix[:,j].argmin(),j]

        if min(dist_matrix[:,j]) < gap_tolerance and s > name_tolerance:
            connectionsA.append(dist_matrix[:,j].argmin())
        else:
            connectionsA.append(-1) # add a value of -1 if it is not joined anything

    for i in range(len(records)):
        if i in connectionsA:
            x = connectionsA.index(i)
            connectionsB.append(x)
        else:
            connectionsB.append(-1) # add a value of -1 if it is not joined anything
#

    return connectionsA, connectionsB


def find_isolated(c1,c2,c3,c4,c5,c6,c7,c8):
    isolated = [] # isolated nodes not joined on either side
    for i in range(len(records)):
        if c1[i]==-1 and c2[i]==-1 and c3[i]==-1 and c4[i]==-1 and c5[i]==-1 and c6[i]==-1 and
            isolated.append(i)
    print 'There are ',len(isolated), 'isolated road segments.'
    return isolated
```

```
        def find_chain(nodei,connection):
                node = nodei
                temp_traversed = []
                while node != -1:
                    node = connection[node]
                    if node != -1 and node not in temp_traversed:
                        chain.append(node)
                        temp_traversed.append(node)
                    else:
                        break
                return chain


        # add attributes and create new shapefile containing road_id attribute
        def create_new_shape(road_id):
            # Read in our existing shapefile
            r = shapefile.Reader("NW_test/NW_test_cleaned")

            # Create a new shapefile in memory
            w = shapefile.Writer()

            # Copy over the existing fields
            w.fields = list(r.fields)

            # Add our new field using the pyshp API
            w.field("road_id", "C", "40")

            x=0
            # Loop through each record, add a column.
            for rec in r.records():
                rec.append(road_id[x][0])
                x+=1
                # Add the modified record to the new shapefile
                w.records.append(rec)

            # Copy over the geometry without any changes
            w._shapes.extend(r.shapes())

            # Save as a new shapefile (or write over the old one)
            w.save("Harrison_Durbin_GISENG_Test_Solution_REVISED")

            return
```

In [4]:
```
# dist_matrix1 = create_dist_matrix(records,name_match_matrix,coords,1)
# dist_matrix2 = create_dist_matrix(records,name_match_matrix,coords,2)
# dist_matrix3 = create_dist_matrix(records,name_match_matrix,coords,3)
# dist_matrix4 = create_dist_matrix(records,name_match_matrix,coords,4)
```

In [5]:
```
records,shapes,coords = load_shape("NW_test/NW_test_cleaned")
print 'Finished loading shapefile.'
```

```
Finished loading shapefile.
```

```
In [6]: name_match_matrix = iter_loadtxt('name_matrix.csv') # load the name_match_matrix from the csv f
        print 'Finished loading name matching matrix.'
```

```
Finished loading name matching matrix.
```

```
In [7]: # name_match_matrix = np.zeros(shape=(len(records),len(records)))
        # for i in range(len(records)):
        #     for j in range(len(records)):
        #         if records[i]==records[j]:
        #             name_match_matrix[i,j]=100
```

```
In [8]: c11,c12 = link_nodes(records,name_match_matrix,'dist_matrix1.csv')
        print 'Finished finding nearest road segments 1.'
```

```
Finished finding nearest road segments 1.
```

```
In [9]: c21,c22 = link_nodes(records,name_match_matrix,'dist_matrix2.csv')
        print 'Finished finding nearest road segments 2.'
```

```
Finished finding nearest road segments 2.
```

```
In [10]: c31,c32 = link_nodes(records,name_match_matrix,'dist_matrix3.csv')
         print 'Finished finding nearest road segments 3.'
```

```
Finished finding nearest road segments 3.
```

```
In [11]: c41,c42 = link_nodes(records,name_match_matrix,'dist_matrix4.csv')
         print 'Finished finding nearest road segments 4.'
```

```
Finished finding nearest road segments 4.
```

```
In [12]: isolated = find_isolated(c11,c12,c21,c22,c31,c32,c41,c42)
```

```
There are  196 isolated road segments.
```

```
In [13]: # # creating chains of nodes and then assigning a unique road id
         x={}
         for i in range(len(records)):
             nodei=i
             if nodei not in isolated:
                 chain = [] # list of nodes that are linked with current node
                 chain.append(i)

                 chain1 = find_chain(nodei,c11)
                 chain2 = find_chain(nodei,c21)
                 chain3 = find_chain(nodei,c31)
                 chain4 = find_chain(nodei,c41)
                 chain5 = find_chain(nodei,c12)
                 chain6 = find_chain(nodei,c22)
                 chain7 = find_chain(nodei,c32)
                 chain8 = find_chain(nodei,c42)

                 # tabulating the unique road ids within the chain
                 chain = np.unique(chain1+chain2+chain3+chain4+chain5+chain6+chain7+chain8)
                 list(chain)
                 x[i]=chain

         print 'Finished combining street chain nodes.'
```

```
Finished combining street chain nodes.
```

```
In [14]: traversed = []
         road_id = np.zeros(shape=(len(records),1))
         road_id[:,:] = -1
         counter = 1

         for i in x:
             chain = x[i]
             chain = list(chain)
             chain.append(i)

             for j in chain:
                 if road_id[j]==-1:
                     if j not in traversed:
                         traversed.append(j)
                         road_id[j] = counter

             counter = 1+max(np.unique(road_id))

         print 'Finished assigning road_id for non-isolated roads.'
```

```
Finished assigning road_id for non-isolated roads.
```

```
In [15]: # assigning unique road ids for any non-linked roads
         unique_ids = np.unique(road_id)

         counter = 1+max(np.unique(road_id))

         for i in range(len(records)):
             if road_id[i]==-1:
                 road_id[i]=counter
             counter += 1

         print 'Finished assigning road_id for isolated roads.'
```

```
Finished assigning road_id for isolated roads.
```

```
In [16]: create_new_shape(road_id)
         print 'Finished creating new shapefile.'
```

```
Finished creating new shapefile.
```

```
In [ ]:
```