UIC Weight and Veterans' Environments Study

Park accessibility: Weight and Veterans' Environments Study GIS protocol

Xiang W, Jones KK, Matthews SA, Zenk SN.

UIC Neighborhoods + Health

Overview

This protocol describes the process by which Navteq and TeleAtlas data were merged to create one comprehensive parks dataset.

Acknowledgements

This protocol was developed with grant support from the National Cancer Institute (R21CA195543) and the Department of Veterans Affairs (IIR 13-085), co-led by Shannon Zenk and Elizabeth Tarlov. We thank Sandy Slater for input into this protocol. Haytham Abu Zayd, Shubhayan Ukil and Abby Klemp helped to edit the protocol for dissemination.

Suggested Citation

Xiang W, Jones K, Matthews SA, Zenk SN. (2018). Park accessibility: Weight and Veterans' Environments Study GIS protocol, Version 1. Retrieved from Weight and Veterans' Environments Study website: https://waves.uic.edu/.

Table of Contents

Background

This document describes the work process for constructing comprehensive park measures. The research team needs to construct a single U.S. national dataset of park polygons. The research team has park polygon files from two sources – NAVTEQ (N) from ESRI StreetMap Premium, and TeleAtlas (T) from ArcGIS Data and Maps. Preliminary analysis shows that either one of the two datasets identifies parks that remain unidentified in the other dataset. The decision was made to merge N and T together based on the rules described in this procedure, to generate a final complete park polygon file.

The general idea of the merging process is to first overlay N and T, (1) if overlapping park polygons can be identified as different parks, then keep both N and T; (2) if overlapping park polygons can be identified as the same park, only keep one (N or T) of the two; (3) for the rest of the parks that are unique to only one dataset in the overlay result, add to the final datasets.

Data

Sources

The research team has park polygon file from two sources – NAVTEQ (N) from ESRI StreetMap Premium, and TeleAtlas (T) from ArcGIS Data and Maps.

Definitions

Park definition and years

The park dataset from N is a feature extraction from the land use file, and park dataset from T is from a park file defined by ESRI. Details of the data can be found in the *Appendix*. The following are a summary of the park categories in each dataset.

The N categories are: Animal Park, Beach, Park (City/County), Park (State), Park in Water, Park/Monument (National), Golf Courses.

The T categories are: National parks or forests, State parks or forests, County parks, Regional parks, Local parks, Golf Courses.

The research team agrees to define park for both datasets as four categories

1. National Parks
2. State Parks
3. Other Parks
4. Golf Courses

This result in park files from N and T as

Final Park Category	**N Source**	**T Source**
National Parks	Park/Monument (National)	National parks or forests
State Parks	Park (State)	State parks or forests
Other Parks	Animal Park, Beach, Park (City/County), Park in Water	County parks, Regional parks, Local parks
Golf Courses	Golf Courses	Golf Courses

Cleaning

The cleaning process uses a set of python scripts to first split all polygons into single polygon (i.e., multi-part features are split into separate features), and then apply a "name matching" procedure to join nearby polygons together if they have same/similar names.

The python script for this part of the process is given in Appendix, Script A. When determining matching score threshold, recommended to use manual matching scores that are between 200 and 262, those above 262 will be automatically matched.

Decisions

Merge rule construction process

The merge rules used were developed after careful analysis of three test cities: Chicago, Boston, and Denver. It was then confirmed/validated using Houston (urban), Bakersfield (smaller city), and Potter County, Pennsylvania (rural). This included sensitivity testing based on park size.

We include here a general description of the process, followed by a pseudocode. The python code used to complete the process is included in the appendix.

Using the Tabulate Intersection tool run twice, switching the zone and class inputs, we identified park polygons in each dataset that were partially, nearly completely, and not at all overlapping with the other dataset. For polygons that overlapped between greater than 20% for both Tabulate Intersection runs, one is chosen based on a set of selection criteria making use of the overlap in each Tabulate Intersection run. For all other cases, both polygons are retained in the final dataset. **.**

Automate the merging process using Python

Python scripts built using ArcPy was developed to automate the process. The scripts are given in appendix Script B.

A pseudocode of the process is constructed as below.

Python pseudocode

The general rule is if the overlay means different parks, then we keep both; if the overlay means the same parks, we ONLY keep 1

Define variables:
N = PCT_OVERLAY_IN_N (percent of TeleAtlas polygon in Navteq)
T = PCT_OVERLAY_IN_T (percent of Navteq polygon in TeleAtlas) (variables returned from Tabulate Intersection tool)

Define functions:

```
def pickOneA() :
    Control under each case: For each selected N or T, if the T or N in this pair was already selected in
    previous step, remove it from the selection. Also generate a list of NOT selected features in N and T.
   case 1: if (N>80 and T>80 and N>T), Pick T
   case 2: if (N>80 and T<80 and N>T), Pick T
   case 3: if (N<80 and T<80 and N>T), Pick T or N
   case 4: if (N>80 and T>80 and N<T), Pick N
   case 5: if (N<80 and T<80 and N<T), Pick T or N
    case 6: if (N<80 and T>80 and N<T), Pick N

def pickOneB():
    Control: For each pair of N, T, if N or T is not in the NOT selected features, select either N or T with
    the larger area. For each selected N or T, if the T or N in this pair was already selected in previous
    step, remove it from the selection. Also add NOT selected feature to the existing list.
    if (N<T):
       pick N
    else:
        pick T

def pickTwo()
    Control: Generate a list of selected features in N, T
def pickStandAlone_N()
def pickStandAlone_T()
```

Code flow:

```
for each N,T pair in N,T total sample: #step 1#
     if (N<20 AND T<20):
        pickTwo()
          remove these samples from the total sample
for each N,T pair in rest sample:     #step 2#
     if (N>60 AND T>60):
        pickOneA()
        remove these samples from the rest sample
for each N, T pair in rest sample: #step 3#
     if (N>80 OR T>80):
         pickOneB()
         remove these samples from the rest sample
for each N, T pair in rest sample: #step 4#
     if (N>20 AND T>20):
         pickOneB()
        remove these samples from the rest sample
for each N, T pair in rest sample: #step 5#
        pickTwo()
for each N, T pair that do not overlay with each other: : #step 6#
          pickStandAlone_N()
          pickStandAlone_T()
```

Park raster grids generation process

After the merge process, the final park dataset will include the following four categories:

National Park

State Park

Other Park

Golf Courses

To generate park measurement as one of the environment measures in the study, the group decided that for the above park categories, except for Golf Courses, generate park grid measures for three distances (400m, 1600m, and 4800m) for all the other three park categories. For this, the total area of park space or number of different parks within the given distance of each cell centroid was calculated.

The final park grid measures will consist of the following two sets for National Park, State Park, Other Park:

Park count grids

Park area grids

The park count grids are generated by first creating a raster of the merged park polygons (via Polygon to Raster tool, using the OBJECTID field as the value field, so that each park will be represented by cells of the unique value), then run the Focal Statistics tool on the newly created raster file with VARIETY as the statistics type. The values in the result park grid represent the number of unique park count.

The park area grids are generated by first creating another raster of the merged park polygons (via Polygon to Raster tool, using COUNT field (COUNT field has been added to this merged park file during the parks merging process) as the value field, so that any place that has park presence will have cell value of 1, no park will have cell value of 0), then run the Focal Statistics with SUM as the statistics type. The values in the result park grid represent the park area in the form of cell count. 1 cell is 30x30 meters, so the park area can be calculated as CELL_VALUE x 900 square meters.

A Python script was used to generate the park grids which is given in Appendix, Script C.

Appendix

Complete Park Data Inventory

This is a complete park data inventory from 2 vendors – TeleAtlas (now subsidiary of TomTom) and N (now subsidiary of HERE). Both vendors partner with ESRI to ship their data. TeleAtlas data comes with Data and Maps for ArcGIS (which ships in DVD with the purchase of ArcGIS software); N data comes with the additional purchase of StreetMap Premium for ArcGIS DVD.

File Name	Content	Year	Source
TELEATLAS SOURCE			
Parks.sdc	National Park Service Land(D83), State and local parks and forests(D85), Local Park or Recreation Area(D89)	**2009** No "Golf Courses" data in this set	TeleAtlas/TomTom
parks_dtl.sdc	National parks or forests, State parks or forests, County parks, Regional parks, Local parks	**2010**	TeleAtlas/TomTom
lalndmrk.sdc (feature extraction)	Amusement Parks, Golf Courses, Stadiums	**2010**	TeleAtlas/TomTom
parks_dtl.sdc	National parks or forests, State parks or forests, County parks, Regional parks, Local parks	**2011**	TeleAtlas/TomTom
lalndmrk.sdc (feature extraction)	Amusement Parks, Golf Courses, Stadiums	**2011**	TeleAtlas/TomTom
park_dtl.sdc	National park or forest, State park or forest, County park, Regional park, Local park	**2012**	TeleAtlas/TomTom
lalndmrk.sdc (feature extraction)	Amusement Parks, Golf Courses, Stadiums	**2012**	TeleAtlas/TomTom
park_dtl	National park or forest, State park or forest, County park, Regional park, Local park	**2014**	TeleAtlas/TomTom
lalndmrk (feature extraction)	Amusement Parks, Golf Courses, Stadiums	**2014**	TeleAtlas/TomTom
LandUseA.shp in 9 folders (feature extraction)	Animal Park, Beach, Park (City/County), Park (State), Park in Water, Park/Monument (National)	**2010**	N/HERE

LandUseA.shp and LandUseB.shp in 9 folders (feature extraction)	AmusementParks (LandUseA), GolfCourses (LandUseB), SportsComplexes (LandUseA)	**2010**	N/HERE
landusea.sdc (feature extraction)	Animal Park, Beach, Park (City/County), Park (State), Park in Water, Park/Monument (National)	**2013**	N/HERE
landusea.sdc and landuseb.sdc (feature extraction)	AmusementParks (landusea), GolfCourses (landuseb), SportsComplexes (landusea)	**2013**	N/HERE
MapLandArea (feature extraction, "FEAT_TYPE")	2000461(Animal Park), 509998(Beach), 900150(Park (City/County)), 900130(Park (State)), 900140(Park in Water), 900103(Park/Monument (National))	**2014**	N/HERE
MapLandArea (feature extraction, "FEAT_TYPE")	2000460(AMUSEMENT PARK), 2000123(GOLF COURSE), 2000457(SPORTS COMPLEX)	**2014**	N/HERE

Scripts

A: Split/Dissolve

```
'''
FIRST SCRIPT IN SPLIT/DISSOLVE PROCESS
'''

import arcpy, sys, os, csv
from arcpy import env

# Set workspace environment
env.workspace = sys.argv[1]
print "sys.argv[1] is "+ sys.argv[1]
# Allow overwrite within workspace
env.overwriteOutput = True

splitInFeatureClass = sys.argv[2]
splitOutFeatureClass = sys.argv[2] + "_singlepart"

# Split multipart features using the ArcGIS Multipart To Singlepart tool
arcpy.MultipartToSinglepart_management(splitInFeatureClass, splitOutFeatureClass)

# Find candidates near each feature using the ArcGIS Generate Near Table tool

# Set Near Table parameters
in_features = splitOutFeatureClass
near_features = splitOutFeatureClass
out_table = splitInFeatureClass + "_nearTable"
search_radius = sys.argv[3] + ' Meters'
location = 'NO_LOCATION'
angle = 'NO_ANGLE'
closest = 'ALL'
closest_count = int(sys.argv[4])

# Generate Near Table
arcpy.GenerateNearTable_analysis(in_features, near_features, out_table, search_radius, location, angle,
closest, closest_count)

# Add index to target feature field prior to join
arcpy.AddIndex_management (out_table, "IN_FID", "IN_FIDIndex", "NON_UNIQUE", "ASCENDING")

# Add index to candidate feature field prior to join
arcpy.AddIndex_management (out_table, "NEAR_FID", "NEAR_FIDIndex", "NON_UNIQUE",
"NON_ASCENDING")

# # Join original table attributes of the target feature to the Near Table
arcpy.JoinField_management(out_table, "IN_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5],
"ORIG_FID"])
```

```
# # Join original table attributes of the candidate feature to the Near Table
arcpy.JoinField_management(out_table, "NEAR_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5]])

# Convert the table to a csv file, and export it to the working directory
def exportToCSV(in_table, out_table, field_names):
        with open(out_table,'wb') as f:
                dw = csv.DictWriter(f,field_names)
                dw.writeheader()
                with arcpy.da.SearchCursor(in_table,field_names) as cursor:
                        for row in cursor:
                                dw.writerow(dict(zip(field_names,row)))

in_table = out_table
out_table = sys.argv[6] + out_table + ".csv"

fields = arcpy.ListFields(in_table)
field_names = [field.name for field in fields]
print field_names
exportToCSV(in_table, out_table, field_names)

# Convert the feature class attribute table that is the result of the split multipart features process in this script to a csv file, to be used
# in the fourth script (dissolve common IDs).

fields = arcpy.ListFields(splitOutFeatureClass)
field_names = [field.name for field in fields]
objectid = [field_names[0]]

exportToCSV(splitOutFeatureClass, sys.argv[6] + splitOutFeatureClass + ".csv", objectid)
```

B: Merge Scripts

A1: Merge

```
'''
This script uses the joined table (which has 2 percent values), to merge the 2 datasets - NAVTEQ and TeleAtlas together as one, based on the rules defined below.
'''

import arcpy, sys, os, time
import itertools

start = time.time()
print "starting time is ....."
print start

#set up workspace
in_workspace = sys.argv[1]
input_N = sys.argv[2]
input_T = sys.argv[3]
arcpy.env.workspace = in_workspace
arcpy.env.overwriteOutput = True

#locate the NAVTEQ and TeleAtlas feature classes
features = arcpy.ListFeatureClasses()
#the NAVTEQ and TeleAtlas features classes name are "_postSplitDissolve_FINAL_OUTPUT"
for feature in features:
        if feature.find(input_N) != -1:
                feature_N = feature
        else:
                feature_T = feature

#define layer file
layer_N = "N_lyr"
layer_T = "T_lyr"


#make layer file for N feature classes (SelectLayerByAttribute only works on lyr file)
arcpy.MakeFeatureLayer_management(feature_N, layer_N)
#add a field "orig_objectid" to keep track of the original OBJECTID of N
arcpy.AddField_management(layer_N, "orig_objectid", "SHORT", "", "", "", "", "NULLABLE", "REQUIRED")
#add a field "Source" to indicate from N or T the polygon is from
arcpy.AddField_management(layer_N, "Source", "TEXT", "", "", "", "", "NULLABLE", "REQUIRED")
#add a field "count" for the purpose of constructing park area, 1 indicates park existence
arcpy.AddField_management(layer_N, "count", "SHORT", "", "", "", "", "NULLABLE", "REQUIRED")
arcpy.CalculateField_management(layer_N, "orig_objectid", '[OBJECTID]')
arcpy.CalculateField_management(layer_N, "Source", '"N"')
arcpy.CalculateField_management(layer_N, "count", '1')

#make layer file for T feature classes (SelectLayerByAttribute only works on lyr file)
arcpy.MakeFeatureLayer_management(feature_T, layer_T)
#add a field "orig_objectid" to keep track of the original OBJECTID of T
arcpy.AddField_management(layer_T, "orig_objectid", "SHORT", "", "", "", "", "NULLABLE", "REQUIRED")
#add a field "Source" to indicate from N or T the polygon is from
arcpy.AddField_management(layer_T, "Source", "TEXT", "", "", "", "", "NULLABLE", "REQUIRED")
#add a field "count" for the purpose of constructing park area, 1 indicates park existence
arcpy.AddField_management(layer_T, "count", "SHORT", "", "", "", "", "NULLABLE", "REQUIRED")
arcpy.CalculateField_management(layer_T, "orig_objectid", '[OBJECTID]')
arcpy.CalculateField_management(layer_T, "Source", '"T"')
arcpy.CalculateField_management(layer_T, "count", '1')

# in Step 2, keep track of IDs that are not selected from N, T selection
```

```
global notSelectedID_N
notSelectedID_N = []
global notSelectedID_T
notSelectedID_T = []

#This function picks feature from N dataset that does not overlay with T
def pickStandAlone_N(id):
        where = '"OBJECTID" = ' + str(id)
        arcpy.SelectLayerByAttribute_management(layer_N, "ADD_TO_SELECTION", where)


#This function picks feature from T dataset that does not overlay with N
def pickStandAlone_T(id):
        where = '"OBJECTID" = ' + str(id)
        arcpy.SelectLayerByAttribute_management(layer_T, "ADD_TO_SELECTION", where)

#This function choose both N and T from the overlay
def pickTwo(idN,idT):
        if idN not in notSelectedID_N:
                where_N = '"OBJECTID" = ' + str(idN)
                arcpy.SelectLayerByAttribute_management(layer_N,
"ADD_TO_SELECTION", where_N)
        if idT not in notSelectedID_T:
                where_T = '"OBJECTID" = ' + str(idT)
                arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)


#Pick one out of the two parks based on H, L relationship
def pickOneA(idN,idT,N,T):
        #Define 60<L<80, H>80
        #case 1 - H>H
        if(N>80 and T>80 and N>T):
                #Pick T
                where_T = '"OBJECTID" = ' + str(idT)
                #if N in this pair was selected in Step 1, need to remove it from N
selection
                where_N = '"OBJECTID" = ' + str(idN)
                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                #if N in this pair was already selected in Step 1, remove it
                if idN in selected_N:
                        arcpy.SelectLayerByAttribute_management(layer_N,
"REMOVE_FROM_SELECTION", where_N)
                #select T
                arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)
                #add the non-selected idN to the list
                notSelectedID_N.append(idN)
        #case 2 - H>L
        if(N>80 and T<80 and N>T):
                #Pick T
                where_T = '"OBJECTID" = ' + str(idT)
                #if N in this pair was selected in Step 1, need to remove it from N
selection
                where_N = '"OBJECTID" = ' + str(idN)
                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                #if N in this pair was already selected
                if idN in selected_N:
                        arcpy.SelectLayerByAttribute_management(layer_N,
"REMOVE_FROM_SELECTION", where_N)
                #selected T
                arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)
                #add the non-selected idN to the list
                notSelectedID_N.append(idN)
        #case 3 - L>L
        if(N<80 and T<80 and N>T):
                #Pick T or N - select T
                where_T = '"OBJECTID" = ' + str(idT)
                #if N in this pair was selected in Step 1, need to remove it from N
selection
                where_N = '"OBJECTID" = ' + str(idN)
                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                #if N in this pair was already selected
                if idN in selected_N:
                        arcpy.SelectLayerByAttribute_management(layer_N,
"REMOVE_FROM_SELECTION", where_N)
                #select T
                arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)
                #add the non-selected idN to the list
                notSelectedID_N.append(idN)
        #case 4 - H<H
        if(N>80 and T>80 and N<T):
                #Pick N
                where_N = '"OBJECTID" = ' + str(idN)
```

```
                                #if T in this pair was selected in Step 1, need to remove it from T
selection
                                where_T = '"OBJECTID" = ' + str(idT)
                                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                                #if T in this pair was already selected
                                if idT in selected_T:
                                                arcpy.SelectLayerByAttribute_management(layer_T,
"REMOVE_FROM_SELECTION", where_T)
                                #select T
                                arcpy.SelectLayerByAttribute_management(layer_N,
"ADD_TO_SELECTION", where_N)
                                #add the non-selected idN to the list
                                notSelectedID_T.append(idT)
                #case 5 - L<L
                if(N<80 and T<80 and N<T):
                                #Pick T or N - select T
                                where_T = '"OBJECTID" = ' + str(idT)
                                #if N in this pair was selected in Step 1, need to remove it from N
selection
                                where_N = '"OBJECTID" = ' + str(idN)
                                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                                #if N in this pair was already selected
                                if idN in selected_N:
                                                arcpy.SelectLayerByAttribute_management(layer_N,
"REMOVE_FROM_SELECTION", where_N)
                                #select T
                                arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)
                                #add the non-selected idN to the list
                                notSelectedID_N.append(idN)
                #case 6 - L<H
                if(N<80 and T>80 and N<T):
                                #Pick N
                                where_N = '"OBJECTID" = ' + str(idN)
                                #if T in this pair was selected in Step 1, need to remove it from T
selection
                                where_T = '"OBJECTID" = ' + str(idT)
                                selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                                selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                                #if T in this pair was already selected
```

```
                    if idT in selected_T:
                              arcpy.SelectLayerByAttribute_management(layer_T,
"REMOVE_FROM_SELECTION", where_T)
                    #select N
                    arcpy.SelectLayerByAttribute_management(layer_N,
"ADD_TO_SELECTION", where_N)
                    #add the non-selected idN to the list
                    notSelectedID_T.append(idT)

#Pick one out of the two parks that has larger area
def pickOneB(idN,idT,N,T):
          #N<T means N area is larger than T, pick N
          if(N<T):
                    if idN not in notSelectedID_N:
                              where_N = '"OBJECTID" = ' + str(idN)
                              #if T in this pair was selected previous steps, need to
remove it from T selection
                              where_T = '"OBJECTID" = ' + str(idT)
                              selected_T = [row[0] for row in
arcpy.da.SearchCursor(layer_T,["OID@"])]
                              #if T in this pair was already selected
                              if idT in selected_T:

     arcpy.SelectLayerByAttribute_management(layer_T, "REMOVE_FROM_SELECTION", where_T)
                              #select N
                              arcpy.SelectLayerByAttribute_management(layer_N,
"ADD_TO_SELECTION", where_N)
                              notSelectedID_T.append(idT)
          else:
                    if idT not in notSelectedID_T:
                              where_T = '"OBJECTID" = ' + str(idT)
                              #if N in this pair was selected in previous steps, need to
remove it from N selection
                              where_N = '"OBJECTID" = ' + str(idN)
                              selected_N = [row[0] for row in
arcpy.da.SearchCursor(layer_N,["OID@"])]
                              #if N in this pair was already selected
                              if idN in selected_N:

     arcpy.SelectLayerByAttribute_management(layer_N, "REMOVE_FROM_SELECTION", where_N)
                              #select T
                              arcpy.SelectLayerByAttribute_management(layer_T,
"ADD_TO_SELECTION", where_T)
                              notSelectedID_N.append(idN)
```

```
#locate the table
tableList = arcpy.ListTables()
for table in tableList:
        #the table name is
"NAVTEQ_YEAR_Tele_YEAR_tabulate_Tele_YEAR_NAVTEQ_YEAR_tabulateJOINED"
        #so locate the table which contains the word "JOINED"
        if table.find("JOINED") != -1:
                outTable = table

#array to keep track of examined row in the table
id_table_examined = []
# get the total number of rows count
row_count = arcpy.GetCount_management(outTable)
# Step 1 - If both N, T are smaller than 20% - indicates different parks - PICK TWO
for row in arcpy.da.SearchCursor(outTable,"*"):
        id_table = row[0]
        id_N = row[1]
        id_T = row[2]
        N = row[3]
        T = row[4]
        if (N<20 and T<20):
            id_table_examined.append(id_table)
            pickTwo(id_N,id_T)


pct = round(float(len(id_table_examined))/float(row_count.getOutput(0))*100, 1)
msg = "####### STEP 1 - PICK TWO ##########\n"+str(len(id_table_examined))+" out of
"+str(row_count)+ " ("+str(pct)+"%)"+" are with both N and T < 20% - PICK TWO"
print msg+"\n"

# Step 2 - Rows from Step 1 are removed. For the rest, if both N, T are larger than 60% - indicates same
park - PICK ONE

count = 0
for row in arcpy.da.SearchCursor(outTable,"*"):
        id_table = row[0]
        id_N = row[1]
        id_T = row[2]
        N = row[3]
        T = row[4]
        if id_table not in id_table_examined:
                if (N>60 and T>60):
                        id_table_examined.append(id_table)
                        count += 1
                        pickOneA(id_N,id_T,N,T)

pct = round(float(count)/float(row_count.getOutput(0))*100, 1)
```

```
msg = "####### STEP 2 - PICK ONE ##########\n"+str(count)+" out of "+str(row_count)+"
("+str(pct)+"%)"+" are with both N and T > 60% - PICK ONE"
print msg+"\n"



# Step 3 - Rows from Step 1,2 are removed. For the rest, if either N or T is larger than 80% - indicates
same park - PICK ONE
count = 0
for row in arcpy.da.SearchCursor(outTable,"*"):
            id_table = row[0]
            id_N = row[1]
            id_T = row[2]
            N = row[3]
            T = row[4]
            if id_table not in id_table_examined:
                        if (N>80 or T>80):
                                    id_table_examined.append(id_table)
                                    count += 1
                                    pickOneB(id_N,id_T,N,T)

pct = round(float(count)/float(row_count.getOutput(0))*100, 1)
msg = "####### STEP 3 - PICK ONE ##########\n"+str(count)+" out of "+str(row_count)+"
("+str(pct)+"%)"+" are with either N or T > 80% - PICK ONE"
print msg+"\n"


# Step 4 - Rows from Step 1,2,3 are removed. For the rest, if both N and T are larger than 20% - indicates
same park - PICK ONE
count = 0
for row in arcpy.da.SearchCursor(outTable,"*"):
            id_table = row[0]
            id_N = row[1]
            id_T = row[2]
            N = row[3]
            T = row[4]
            if id_table not in id_table_examined:
                        if (N>20 and T>20):
                                    id_table_examined.append(id_table)
                                    count += 1
                                    pickOneB(id_N,id_T,N,T)

pct = round(float(count)/float(row_count.getOutput(0))*100, 1)
msg = "####### STEP 4 - PICK ONE ##########\n"+str(count)+" out of "+str(row_count)+"
("+str(pct)+"%)"+" are with both N and T > 20% - PICK ONE"
print msg+"\n"

# Step 5 - Rows from Step 1,2,3,4 are removed. For everything left - PICK TWO
```

```
count = 0
for row in arcpy.da.SearchCursor(outTable,"*"):
        id_table = row[0]
        id_N = row[1]
        id_T = row[2]
        N = row[3]
        T = row[4]
        if id_table not in id_table_examined:
                id_table_examined.append(id_table)
                count += 1
                pickTwo(id_N,id_T)

pct = round(float(count)/float(row_count.getOutput(0))*100, 1)
msg = "####### STEP 5 - PICK TWO ##########\n"+str(count)+" out of "+str(row_count)+"
("+str(pct)+"%)"+" are left - PICK TWO"
print msg+"\n"

#Step 6 - locate the features from complete N,T that do not overlay with each other, and add them to
the final output of N, TWO
#get all OBJECTID values from N,T
IDValueList_N = [row[0] for row in arcpy.da.SearchCursor(feature_N,["OID@"])]
IDValueList_T = [row[0] for row in arcpy.da.SearchCursor(feature_T,["OID@"])]
#get OBJECTID values from N_T_tabulate_JOINED - overlaid N and T
IDValueList_overlaid_N = [row[1] for row in arcpy.da.SearchCursor(outTable,"*")]
IDValueList_overlaid_T = [row[2] for row in arcpy.da.SearchCursor(outTable,"*")]

#Select those independent features in N, T that do not overlay with each other, add them to final
selection
count_N = 0
for id_N in IDValueList_N:
        if id_N not in IDValueList_overlaid_N:
                pickStandAlone_N(id_N)
                count_N += 1
count_T = 0
for id_T in IDValueList_T:
        if id_T not in IDValueList_overlaid_T:
                pickStandAlone_T(id_T)
                count_T += 1

msg = "####### (LAST STEP) STEP 6 - UNIQUE PARKS IN N OR T - PICK TWO
##########\n"+str(count_N)+" unique parks in N are added.\n"+str(count_T)+" unique parks in T are
added.\n"
print msg


#Generate list of selected features in N,T
'''N_desc = arcpy.Describe(layer_N)
selectedID_N = N_desc.FIDset
```

```
selectedID_N_list = selectedID_N.split("; ")
T_desc = arcpy.Describe(layer_T)
selectedID_T = T_desc.FIDset
selectedID_T_list = selectedID_T.split("; ")
print "SELECTED OBJECTID IN N"
print selectedID_N_list, len(selectedID_N_list)
print "SELECTED OBJECTID IN T"
print selectedID_T_list, len(selectedID_T_list)
'''

#Generate final output for N and T
arcpy.FeatureClassToFeatureClass_conversion(layer_N, in_workspace, input_N+"_final")
arcpy.FeatureClassToFeatureClass_conversion(layer_T, in_workspace, input_T+"_final")


#Merge the final output of N and T to a single feature class
fieldMappings = arcpy.FieldMappings()
fieldMappings.addTable(layer_N)
fieldMappings.addTable(layer_T)
#only keep "orig_objectid","Source" in the final merged dataset
for field in fieldMappings.fields:
                if field.name not in ["orig_objectid","Source","count"]:

        fieldMappings.removeFieldMap(fieldMappings.findFieldMapIndex(field.name))
arcpy.Merge_management([layer_N,layer_T],input_N+input_T+"_FINAL_MERGED",fieldMappings)

end = time.time()
print "end time is....."
print end
print "time elapsed...."
print str(end-start) + " seconds"
```

A2: Temp Merge

```
'''
FIRST SCRIPT IN SPLIT/DISSOLVE PROCESS
'''

import arcpy, sys, os, csv
from arcpy import env

# Set workspace environment
env.workspace = sys.argv[1]
print "sys.argv[1] is "+ sys.argv[1]
# Allow overwrite within workspace
env.overwriteOutput = True

splitInFeatureClass = sys.argv[2]
```

```
splitOutFeatureClass = sys.argv[2] + "_singlepart"

# Split multipart features using the ArcGIS Multipart To Singlepart tool
arcpy.MultipartToSinglepart_management(splitInFeatureClass, splitOutFeatureClass)

# Find candidates near each feature using the ArcGIS Generate Near Table tool

# Set Near Table parameters
in_features = splitOutFeatureClass
near_features = splitOutFeatureClass
out_table = splitInFeatureClass + "_nearTable"
search_radius = sys.argv[3] + ' Meters'
location = 'NO_LOCATION'
angle = 'NO_ANGLE'
closest = 'ALL'
closest_count = int(sys.argv[4])

# Generate Near Table
arcpy.GenerateNearTable_analysis(in_features, near_features, out_table, search_radius, location, angle,
closest, closest_count)

# Add index to target feature field prior to join
arcpy.AddIndex_management (out_table, "IN_FID", "IN_FIDIndex", "NON_UNIQUE", "ASCENDING")

# Add index to candidate feature field prior to join
arcpy.AddIndex_management (out_table, "NEAR_FID", "NEAR_FIDIndex", "NON_UNIQUE",
"NON_ASCENDING")

# # Join original table attributes of the target feature to the Near Table
arcpy.JoinField_management(out_table, "IN_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5],
"ORIG_FID"])

# # Join original table attributes of the candidate feature to the Near Table
arcpy.JoinField_management(out_table, "NEAR_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5]])

# Convert the table to a csv file, and export it to the working directory
def exportToCSV(in_table, out_table, field_names):
	with open(out_table,'wb') as f:
		dw = csv.DictWriter(f,field_names)
		dw.writeheader()
		with arcpy.da.SearchCursor(in_table,field_names) as cursor:
			for row in cursor:
				dw.writerow(dict(zip(field_names,row)))

in_table = out_table
out_table = sys.argv[6] + out_table + ".csv"

fields = arcpy.ListFields(in_table)
```

```
field_names = [field.name for field in fields]
print field_names
exportToCSV(in_table, out_table, field_names)

# Convert the feature class attribute table that is the result of the split multipart features process in this
script to a csv file, to be used
# in the fourth script (dissolve common IDs).

fields = arcpy.ListFields(splitOutFeatureClass)
field_names = [field.name for field in fields]
objectid = [field_names[0]]

exportToCSV(splitOutFeatureClass, sys.argv[6] + splitOutFeatureClass + ".csv", objectid)
```

A3: Tabulation Analysis

```
'''
FIRST SCRIPT IN SPLIT/DISSOLVE PROCESS
'''

import arcpy, sys, os, csv
from arcpy import env

# Set workspace environment
env.workspace = sys.argv[1]
print "sys.argv[1] is "+ sys.argv[1]
# Allow overwrite within workspace
env.overwriteOutput = True

splitInFeatureClass = sys.argv[2]
splitOutFeatureClass = sys.argv[2] + "_singlepart"

# Split multipart features using the ArcGIS Multipart To Singlepart tool
arcpy.MultipartToSinglepart_management(splitInFeatureClass, splitOutFeatureClass)

# Find candidates near each feature using the ArcGIS Generate Near Table tool

# Set Near Table parameters
in_features = splitOutFeatureClass
near_features = splitOutFeatureClass
out_table = splitInFeatureClass + "_nearTable"
search_radius = sys.argv[3] + ' Meters'
location = 'NO_LOCATION'
angle = 'NO_ANGLE'
closest = 'ALL'
closest_count = int(sys.argv[4])
```

```
# Generate Near Table
arcpy.GenerateNearTable_analysis(in_features, near_features, out_table, search_radius, location, angle, 
closest, closest_count)

# Add index to target feature field prior to join
arcpy.AddIndex_management (out_table, "IN_FID", "IN_FIDIndex", "NON_UNIQUE", "ASCENDING")

# Add index to candidate feature field prior to join
arcpy.AddIndex_management (out_table, "NEAR_FID", "NEAR_FIDIndex", "NON_UNIQUE", 
"NON_ASCENDING")

# # Join original table attributes of the target feature to the Near Table
arcpy.JoinField_management(out_table, "IN_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5], 
"ORIG_FID"])

# # Join original table attributes of the candidate feature to the Near Table
arcpy.JoinField_management(out_table, "NEAR_FID", splitOutFeatureClass, "OBJECTID", [sys.argv[5]])

# Convert the table to a csv file, and export it to the working directory
def exportToCSV(in_table, out_table, field_names):
        with open(out_table,'wb') as f:
                dw = csv.DictWriter(f,field_names)
                dw.writeheader()
                with arcpy.da.SearchCursor(in_table,field_names) as cursor:
                        for row in cursor:
                                dw.writerow(dict(zip(field_names,row)))

in_table = out_table
out_table = sys.argv[6] + out_table + ".csv"

fields = arcpy.ListFields(in_table)
field_names = [field.name for field in fields]
print field_names
exportToCSV(in_table, out_table, field_names)

# Convert the feature class attribute table that is the result of the split multipart features process in this 
script to a csv file, to be used
# in the fourth script (dissolve common IDs).

fields = arcpy.ListFields(splitOutFeatureClass)
field_names = [field.name for field in fields]
objectid = [field_names[0]]

exportToCSV(splitOutFeatureClass, sys.argv[6] + splitOutFeatureClass + ".csv", objectid)
```

A4: Tabulation Analysis Temp

```
'''
This is temp script. Add the information of for those "choose 1 out of 2", add category names, park
names from N, T to tabulate joined table.
FIRST SCRIPT IN MERGE.BAT
This script performs tabulate analysis between NAVTEQ and TeleAtlas, then TeleAtlas and NAVTEQ, and
then joins the two tables together as one.
'''


import arcpy, sys, os
import itertools

#set up workspace
in_workspace = sys.argv[1]
arcpy.env.workspace = in_workspace
arcpy.env.overwriteOutput = True

#name of the NAVTEQ name and TeleAtlas name. ONLY need the first part from the complete name
N_name = sys.argv[2]
T_name = sys.argv[3]

#N_feature = N_name
#T_feature = T_name
N_feature = N_name + "_postSplitDissolve_FINAL_OUTPUT"
T_feature = T_name + "_postSplitDissolve_FINAL_OUTPUT"

#define tabulate groups, perform tabulate analysis for tabulate1 first, then tabulate2, to get the two
percent numbers
tabulate1 = [N_feature,T_feature]
tabulate2 = [T_feature, N_feature]
nameGroup1 = [N_name,T_name]
nameGroup2 = [T_name,N_name]

def AddMsgAndPrint(msg, severity=0):
  # Adds a Message (in case this is run as a tool)
  # and also prints the message to the screen (standard output)
  #
  print msg

  # Split the message on \n first, so that if it's multiple lines,
  #  a GPMessage will be added for each line
  try:
    for string in msg.split('\n'):
      # Add appropriate geoprocessing message
      #
      if severity == 0:
        arcpy.AddMessage(string)
        elif severity == 1:
            arcpy.AddWarning(string)
        elif severity == 2:
            arcpy.AddError(string)
    except:
        pass

count = 0
#Run "Tabulate Analysis" between N and T, and then T and N, generate two tables
for feature1, feature2, name1, name2 in itertools.izip(tabulate1,tabulate2,nameGroup1,nameGroup2):
        ## Get Parameters
        zoneFC = feature1
        if count == 0:
                zoneFld =
["OBJECTID","NAVTEQ_2010_singlepart_FEAT_TYPE","NAVTEQ_2010_singlepart_POLYGON_NM"] #
Only allow one field. Only add OBJECTID to the output table
        else:
                zoneFld =
["OBJECTID","NAVTEQ_2014_singlepart_FEATURE_TYPE","NAVTEQ_2014_singlepart_NAME"]
        classFC = feature2
        outTab = in_workspace + name1 + "_" + name2 + "_tabulate"
        if count == 0:
                classFld =
["OBJECTID","NAVTEQ_2014_singlepart_FEATURE_TYPE","NAVTEQ_2014_singlepart_NAME"] #
Optional and only allow one field. Only add OBJECTID to the output table
        else:
                classFld =
["OBJECTID","NAVTEQ_2010_singlepart_FEAT_TYPE","NAVTEQ_2010_singlepart_POLYGON_NM"]
        sum_Fields = ""
        xy_tol = ""
        outUnits = ""

        ## Validate parameters
        # Inputs can only be polygons
        zoneDesc = arcpy.Describe(zoneFC)
        classDesc = arcpy.Describe(classFC)
        if zoneDesc.shapeType != "Polygon" or classDesc.shapeType != "Polygon":
                AddMsgAndPrint("Inputs must be of type polygon.", 2)
                sys.exit()

        # Only one zone field and class field
        '''if zoneFld != "":
                if zoneFld.find(";") > -1 or classFld.find(";") > -1:
                        AddMsgAndPrint("A maximum of one zone and/or class field is allowed.", 2)
                        sys.exit()
'''
        ## Run TI with restricted parameters
        try:
```

```
                    if arcpy.Exists(outTab):
                              arcpy.Delete_management(outTab)
                    arcpy.TabulateIntersection_analysis(zoneFC, zoneFld, classFC, outTab, classFld, 
sum_Fields, xy_tol, outUnits)
          except:
                    arcpy.AddMessage("Tabulate Intersection Failed.")
          AddMsgAndPrint(arcpy.GetMessages(), 0)
          count +=1

#Run "Make Query Table" tool to join the 2 table as one, to get the 2 percent numbers in one table
tableList = arcpy.ListTables()
#variables to store the field list in N_T_tabulate and T_N tabulate
fieldListNT = []
fieldListTN = []

for count, table in enumerate(tableList):
                    fields = arcpy.ListFields(table)
                    for field in fields:
                              if count == 0:
                                        fieldListNT.append(table+"."+field.name)
                              if count == 1:
                                        fieldListTN.append(table+"."+field.name)

# where clause is "N_T_tabulate.ID_N = T_N_tabulate.ID_N AND N_T_tabulate.ID_T = 
T_N_tabulate.ID_T"
where  = fieldListNT[1]+"="+fieldListTN[4]+" AND "+fieldListNT[4]+"="+fieldListTN[1]
#where  = fieldListNT[1]+"="+fieldListTN[2]+" AND "+fieldListNT[2]+"="+fieldListTN[1]
# only need to save ID_N, ID_T, PCT_N, PCT_T in output table
fieldListOut = 
[fieldListNT[1],fieldListNT[4],fieldListNT[8],fieldListTN[8],fieldListNT[2],fieldListNT[5],fieldListNT[3],fieldLi
stNT[6]]
#fieldListOut = [fieldListNT[1],fieldListNT[2],fieldListNT[4],fieldListTN[4]]
for table in tableList:
                    if table.find(N_name) == 0:
                                        N_table = table
                    elif table.find(T_name) == 0:
                                        T_table = table
outTable = N_table+"_"+T_table+"JOINED"

arcpy.MakeQueryTable_management(tableList,outTable,"NO_KEY_FIELD","",fieldListOut,where)
# save temporary output table as a permanent table
arcpy.TableToTable_conversion(outTable,in_workspace,outTable)
```

C: Park Grid generation

```
#park grids generation
```

Note: Script will need to be adjusted with project-specific file names and locations.

```
import arcpy
from arcpy.sa import *

in_workspace = {insert file location}
out_workspace = {insert file location}
arcpy.env.workspace = in_workspace
arcpy.env.extent = "-2493045.0 -1429501.25 2342655.0 1703218.75"
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension("Spatial")



#set focal statistics variables

neighborhood_400 = NbrCircle(400, "MAP")
neighborhood_1600 = NbrCircle(1600, "MAP")
neighborhood_4800 = NbrCircle(4800, "MAP")
neighborhood_8000 = NbrCircle(8000, "MAP")

# national
# Set focal statistics variables
lyr_national = "National_2010_Count"
arcpy.MakeRasterLayer_management("ntnal_2010_ct", lyr_national)
inRaster = lyr_national

# Execute FocalStatistics
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_400, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2014_National_COUNT_400m")
print inRaster
outFocalStatistics = FocalStatistics(inRaster, neighborhood_1600, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2014_National_COUNT_1600m")
print inRaster
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_4800, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2010_National_COUNT_4800m")
print inRaster
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_8000, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_National_AREA_8000m")
print inRaster
'''

# state
# Set focal statistics variables
lyr_state = "State_2010_Count"
arcpy.MakeRasterLayer_management("state_2010_ct", lyr_state)
inRaster = lyr_state
```

```
# Execute FocalStatistics
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_400, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2010_State_COUNT_400m")
print inRaster
outFocalStatistics = FocalStatistics(inRaster, neighborhood_1600, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2010_State_COUNT_1600m")
print inRaster
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_4800, "VARIETY","")
outFocalStatistics.save(out_workspace+"Park_2010_State_COUNT_4800m")
print inRaster
'''
outFocalStatistics = FocalStatistics(inRaster, neighborhood_8000, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_State_AREA_8000m")
print inRaster
'''
'''
# other
# Set focal statistics variables
lyr_other = "Other_2010_Area"
arcpy.MakeRasterLayer_management("other_2010_ar", lyr_other)
inRaster = lyr_other

# Execute FocalStatistics
outFocalStatistics = FocalStatistics(inRaster, neighborhood_400, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_Other_AREA_400m")
print inRaster
outFocalStatistics = FocalStatistics(inRaster, neighborhood_1600, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_Other_AREA_1600m")
print inRaster
outFocalStatistics = FocalStatistics(inRaster, neighborhood_4800, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_Other_AREA_4800m")
print inRaster
outFocalStatistics = FocalStatistics(inRaster, neighborhood_8000, "SUM","")
outFocalStatistics.save(out_workspace+"Park_2010_Other_AREA_8000m")
print inRaster
'''
```

Software

The software used is ArcGIS 10.3.1 and Python 2.7.