GEOG 381 Winter 2017 Assignment 2

Suitability Evaluation

NORTHEASTERN WASHTENAW COUNTY, MICHIGAN, USA

A development plan maximizing land suitability and minimizing ecological impact.

Jiachen Wei (20695056)

University of Waterloo

Wednesday, 15 February 2017

I. Introduction

The Southeastern Michigan Council of Governments (SEMCOG) requested for a development plan maximizing land suitability and minimizing ecological impact in northeastern Washtenaw County. The 1995 land use data (Figure 1) show that 30.75% of the study area is unavailable; 4.67% of the plannable area is developed, 53.36% agricultural, 41.97% natural. SEMCOG proposed changes from the 1995 levels as follows: a decrease by 35% in agriculture, an increase by 30% in high-density development, and a decline by 24% in natural lands.

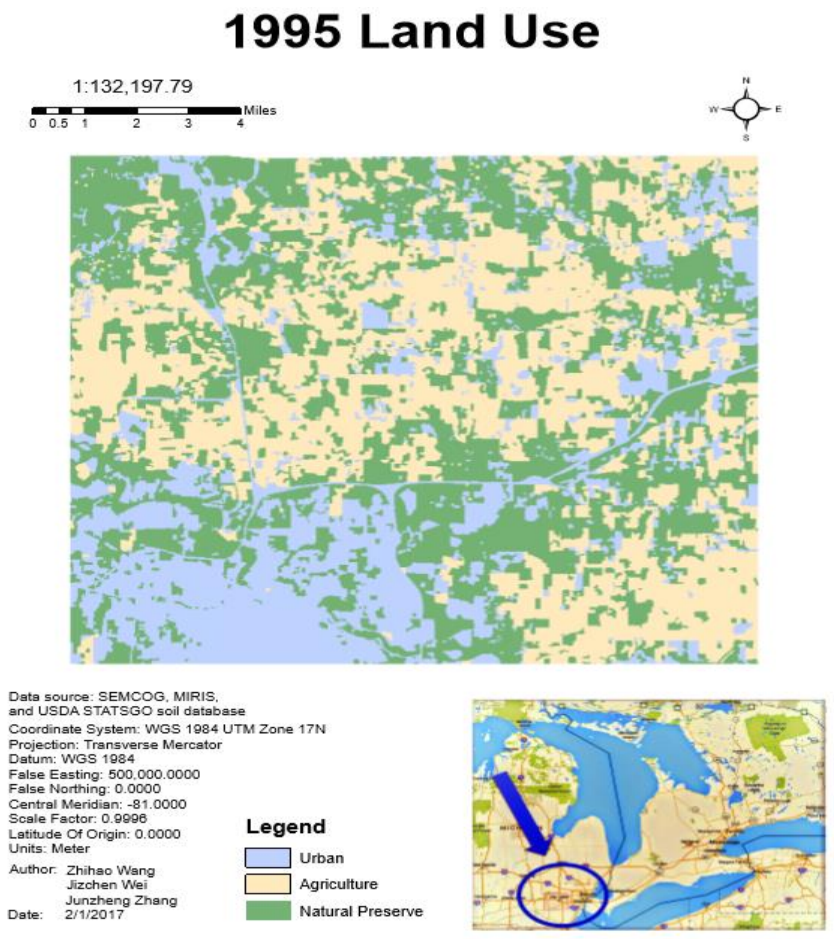
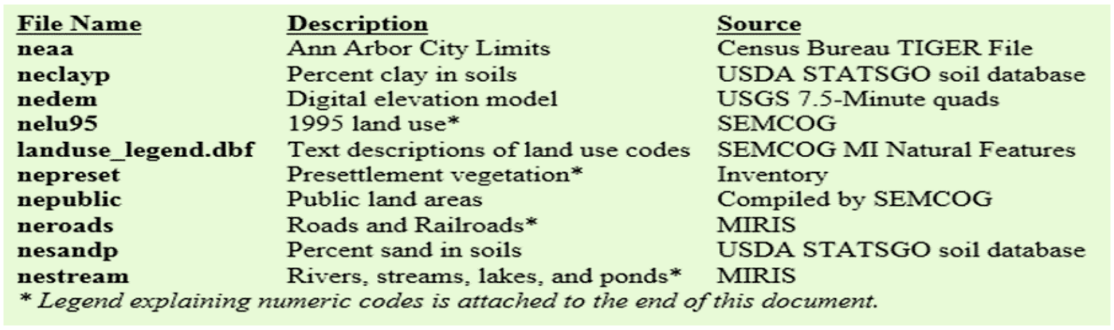


Figure 1 study area

Our spatial analyst team create three suitability maps considering multiple criteria for each of agriculture, development, and nature using ArcGIS Model Builder. Next, we generate a conflict map and a future land-use map in Python's interactive development environment (IDLE). By editing the Python script, we are able to produce better results than using the prewritten ArcGIS tools in terms of validating data, customizing solution, and reducing repetition. This report presents the methods employed in our analysis of conflict and future land use, and discusses the results and limitations.

The data for northeastern Washtenaw County originated from Dr. Daniel Brown at the School of Natural Resources and Environment, University of Michigan. We utilized all the data listed in Table 1, which are all in Arc Grid format at a 30-meter spatial resolution. The land use data (Figure 1) feature an urban area in the northwest corner, two major roads, vast rural areas, and plenty of steams and wetlands.

Table 1 data files



File Name	Description	Source
neaa	Ann Arbor City Limits	Census Bureau TIGER File
neclayp	Percent clay in soils	USDA STATSGO soil database
nedem	Digital elevation model	USGS 7.5-Minute quads
nelu95	1995 land use*	SEMCOG
landuse_legend.dbf	Text descriptions of land use codes	SEMCOG MI Natural Features
nepreset	Presettlement vegetation*	Inventory
nepublic	Public land areas	Compiled by SEMCOG
neroads	Roads and Railroads*	MIRIS
nesandp	Percent sand in soils	USDA STATSGO soil database
nestream	Rivers, streams, lakes, and ponds*	MIRIS

** Legend explaining numeric codes is attached to the end of this document.*

II. Methods

Our team begin with modifying the Python script of the three ArcGIS models, which produce three rasters where plannable lands are assigned suitability values (integers from 1 to 100) for each of development, agriculture, or nature. Next, we code in Python to produce a conflict map, and a future land-use map based on the conflict map.

i. Python Script and Environment

ArcGIS models can be conveniently exported as Python scripts; however, multiple error may occur when running the code in IDLE. We import necessary libraries for later calling functions, set the environment as appropriate for reading-writing and raster size, group blocks of code together following the model diagrams, and create or rename variables in accordance to the parameters used in context. The available IDLE (32-bit for ArcGIS) not being editor-friendly, we edit the scripts in Notepad++ and link them to our ArcGIS toolbox. The scripts are run significantly faster in ArcMap than in IDLE; more importantly, the results can be visualized and better managed in ArcMap.

ii. Creating the Conflict Map

Our team produce a conflict map that shows where more than one criteria are high in suitability scores at the same location. The Python script (Appendix ii.) functions as an ArcGIS tool, requiring the input of raster data and producing the output of one raster map.

Our script first reads the input raster data and convert them to NumPy arrays—a data type in Numerical Python package supporting large arrays and complex operations (ArcMap, 2016). Since the input raster data are integers from 1 to 100, we consider a suitability score greater than 50 units as “a high score”. Therefore, all the cells with “high scores” in each input raster are represented in the output conflict map. To record the conflicting locations, we create a zeros NumPy array *cflCount[row,col]* of the same size as our input raster arrays. We assign one value to a land-use “high scores” in the zeros array, and then add another value when a second “high score” occurs in the same cell (Table 2). This array is finally converted to a raster with values representing how the three types of land use are conflicting, and we can visualize the raster data in ArcMap.

Table 2 designing *cflCount[row,col]*

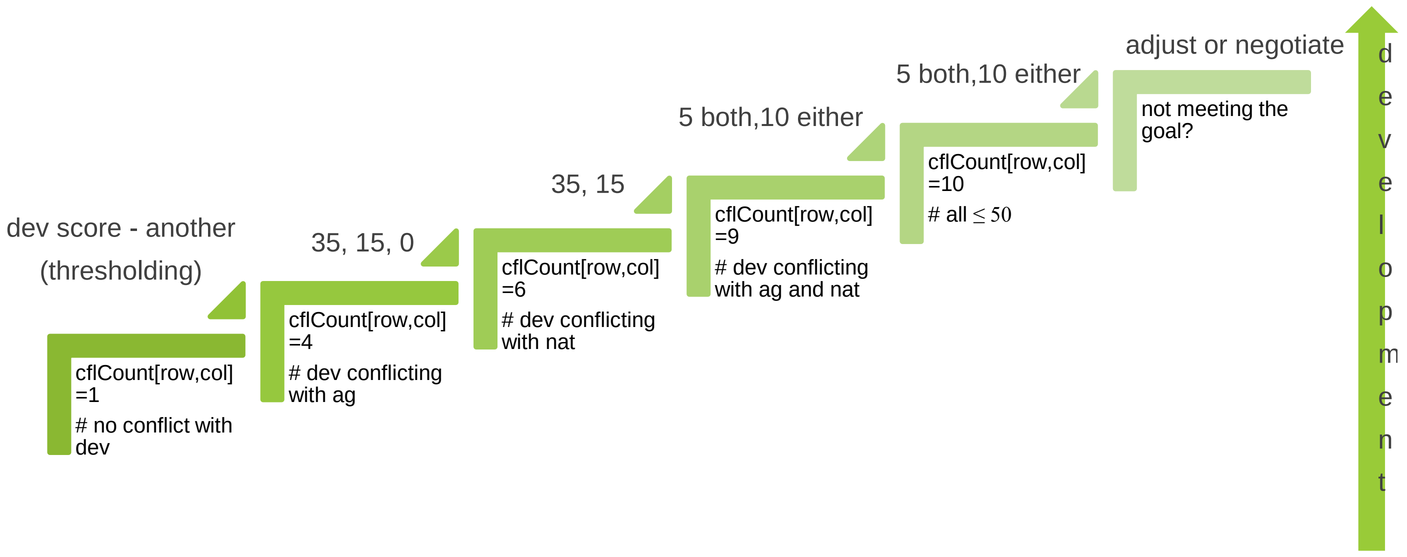
values	
initial value	**0**
high dev score	**1**
high ag score	**3**
high nat score	**5**

final array			
dev only	1	dev+nat	6
ag only	3	ag+nat	8
nat only	5	dev+ag+nat	9
dev+ag	4	all ≤50	10
unavailable land (as the initial value)			0

iii. Producing the Land Use Model

We need to perform a decision rule to allocate land use despite the conflicts. Geographic information systems (GIS) can help to solve the conflicts between land use allocations (Kyem, 2002), and efficiently generate maps to support multi-actor policymaking (Carton, 2007). Our purpose of geocoding in Python is to allow more flexibility in our allocation than relying on ArcGIS tools. The output array (before being converted to raster) is created as a NumPy zeros array named *fluArray*. In the output land-use raster, value 0 represents unavailable lands, 1 development, 2 agriculture, and 3 nature.

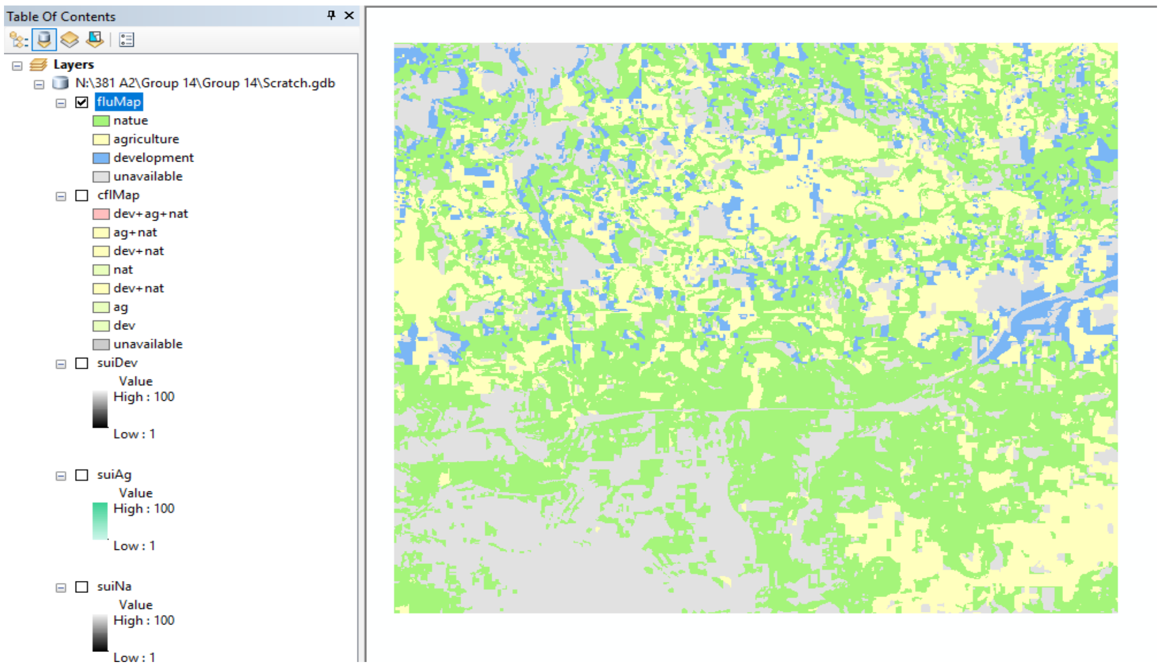
Figure 2 allocating development locations



Based on the conflict map, we follow a conflict resolution rule in our script. As explained in Assignment 1b, the conflict resolution rule allocates one land-use type (development prioritized in our case) first to non-conflicting locations, and second to conflicting locations where its suitability score is the highest. However, with our target percentages in mind, if we still need more locations for development after these two stages of allocation, we can adjust our algorithm or criteria, or negotiate with SEMCOG for a change in their goal (Figure 2). We set up thresholds for conflicting areas so that allocations are first made at locations where development suitability is remarkably higher, and later made at locations with smaller value differences. For example, when dev conflicts with ag, we firstly assign value 1 to the al the elements in *fluArray* when the dev score is over 35 units greater than the ag score; secondly, when the dev score is over 15 units greater (but no greater than 35 units) than the ag score; finally, when the dev score is greater (but no greater than 15 units) than the ag score. This allocation process terminates when the number of 1 values in *fluArray* reaches the development goal (28905, or 10% of the plannable land).

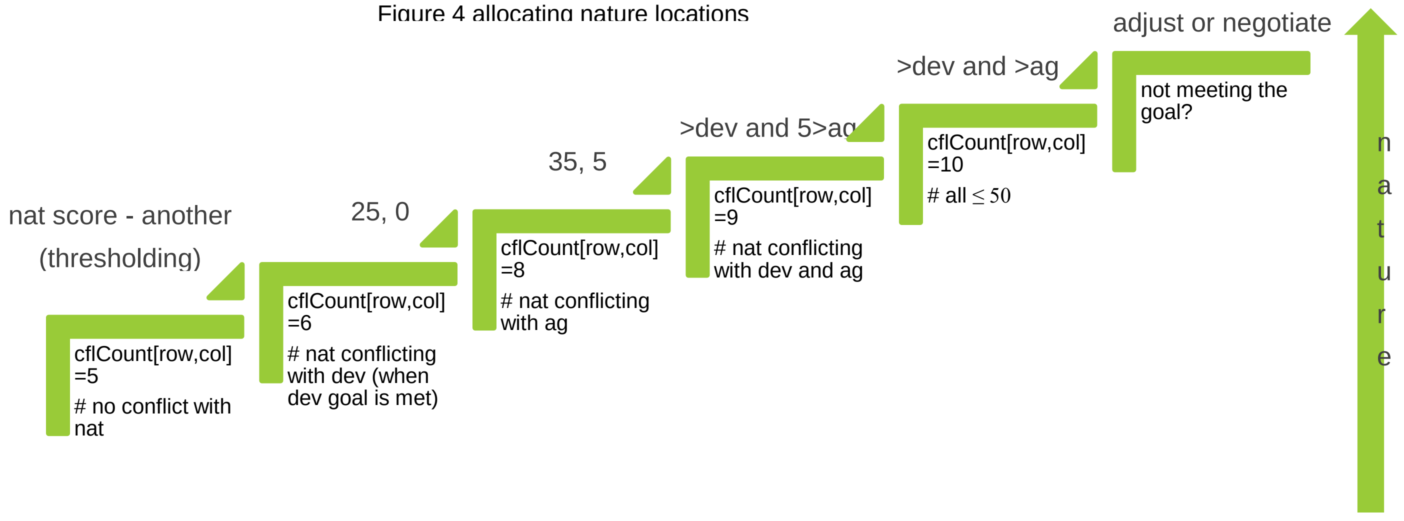
According to Assignment 1b, the conflict decision rule moves on to allocate agriculture land use after meeting the goal for development land use. We perform a similar allocation process to that shown in Figure 2; however, the result is not satisfying. Figure 3 shows the result when the code is run before modification: a number of wetlands are in the center of agricultural lands, which are circled by natural lands (green circles). It makes no logical sense to farm around wetlands and preserve the wetland beyond the "agricultural buffers". In addition, neither dev nor ag meet SEMCOG's goals (respectively about 500 cells and 1000 cells smaller than the target areas). In response to the surplus of nature land use, we modify the thresholds for several times to encourage development "at the cost of nature preservation" and to restrict agriculture by increasing the ag-minus-nat threshold up to 45 units. Unfortunately, the "agricultural buffers" remain while the number of agricultural cells become even smaller.

Figure 3 allocating ag before nat



Alternatively, we allocate nature land use before allocating agriculture locations. Figure 4 illustrates the allocation of nature land use, the idea is the same as mention above: first allocate the non-conflicted locations, and then allocate the higher-value conflicted locations with thresholds. To restrict the "sprawling" of agriculture as mentioned above, the value-difference thresholds are very small (disfavoring agriculture), to an extent comparable to how nature "annexes"(threshold towards 0) the “leftover”(assuming the dev goal is met) development land use. When nature allocation ends, agriculture takes over all the remaining available locations.

Figure 4 allocating nature locations



III. Result and Discussion

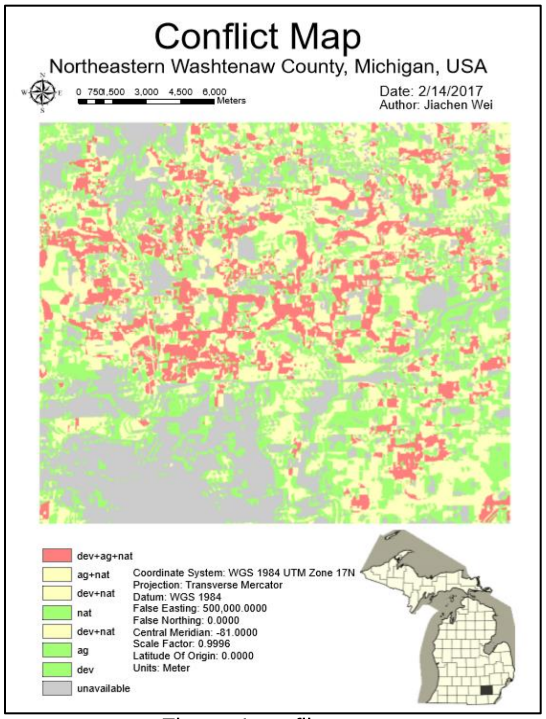
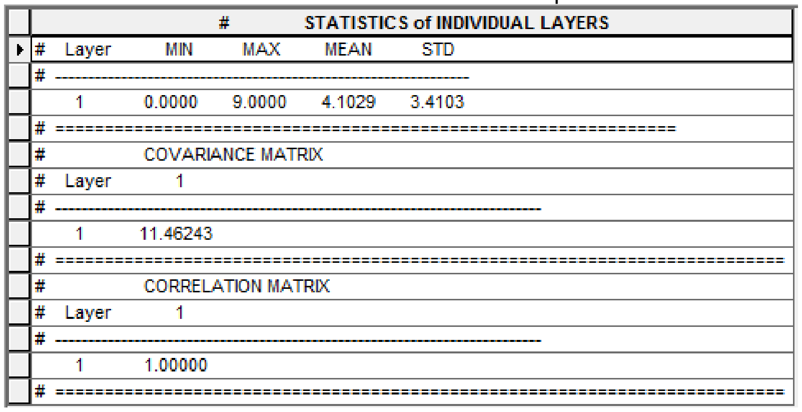

Conflict Map
Northeastern Washtenaw County, Michigan, USA
0 750 1,500 3,000 4,500 6,000 Meters
Date: 2/14/2017
Author: Jiachen Wei
🟥 dev+ag+nat
🟨 ag+nat
🟨 dev+nat
🟩 nat
🟨 dev+nat
🟩 ag
🟩 dev
⬜ unavailable
Coordinate System: WGS 1984 UTM Zone 17N
Projection: Transverse Mercator
Datum: WGS 1984
False Easting: 500,000.0000
False Northing: 0.0000
Central Meridian: -81.0000
Scale Factor: 0.9996
Latitude Of Origin: 0.0000
Units: Meter
Figure 4 conflict map

Figure 4 is the conflict map featuring suitability scores over 50 units for the three types of land use. The green areas indicate “no conflicting land use” (37.94% of the available land), the yellow “two conflicting” (41.95%), and the red “all three conflicting” (20.11%). With reference to 1995 Land Use, the most competitive areas (strips) are located in the northern suburb and towards northeast into the agriculture-dominated areas. Table 3 is the raster statistics generated by the Band Collection Statistics tool. However, given that the values in our conflict map represent additive markers (Table 1) rather than consistent spatial features, statistical measures like mean and standard deviation are not applicable to the interpretation of the data. Were we more forward-looking when designing or producing data, we could analyze the data with such simple statistic approaches.

Table 3 statistics of conflict map

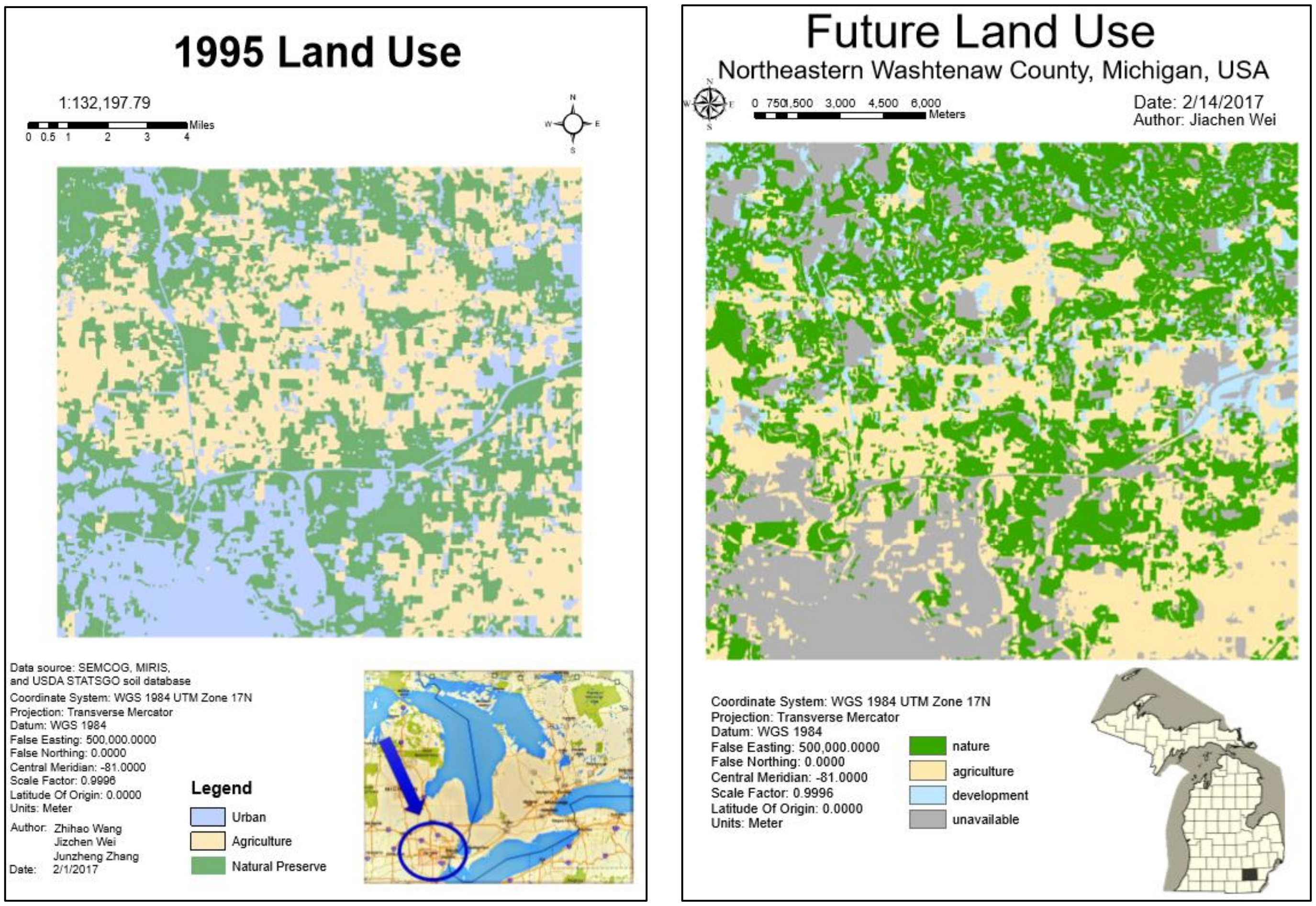


```
#           STATISTICS of INDIVIDUAL LAYERS
# Layer      MIN       MAX       MEAN      STD
# ---------------------------------------------------------------
    1      0.0000    9.0000    4.1029    3.4103
# ================================================================
#                COVARIANCE MATRIX
# Layer          1
# ---------------------------------------------------------------------------
    1       11.46243
# ============================================================================
#                CORRELATION MATRIX
# Layer          1
# ---------------------------------------------------------------------------
    1        1.00000
# ============================================================================
```

The red strips (three-competitor areas) tend to be surrounded by yellow (two-competitor) areas. This suggests a third suitability score suddenly appear high where the other two are dominant. Although basic statistics may not help us interpret the data, we can analyze these area objects in terms of adjacency and autocorrelation (O'Sullivan & Unwin, 2010). Moreover, we can change the color of the two-competitor areas to visually investigate why a third competitor gains advantage in a specific area (with reference to our criteria and weighting scheme).

Land use change has practically met SEMCOG’s goal. Of the 289052 units of plannable land in 1995, 10% (28905 units) are aimed for development, 36% (104059 units) for agriculture, and 54% (156088 units) for nature preserve. In our future land use map (Figure 5), of all the plannable land, 9.07% (28486 units) are for development, 41.92% (131574 units) for agriculture, and 49.01% (153833 units) for nature. A decline in agriculture land use is evident as proposed, natural land expands and interconnects in the northeastern corner of the study area, and development can be prioritized in the east—a blue aggregate on the right-hand side in the map, split by a major road and surrounded by agricultural land.

Figure 5 land use change


1995 Land Use
1:132,197.79
0 0.5 1 2 3 4 Miles
N
W E
S
Data source: SEMCOG, MIRIS, and USDA STATSGO soil database
Coordinate System: WGS 1984 UTM Zone 17N
Projection: Transverse Mercator
Datum: WGS 1984
False Easting: 500,000.0000
False Northing: 0.0000
Central Meridian: -81.0000
Scale Factor: 0.9996
Latitude Of Origin: 0.0000
Units: Meter
Author: Zhihao Wang Jizchen Wei Junzheng Zhang
Date: 2/1/2017
Legend
Urban
Agriculture
Natural Preserve
Future Land Use
Northeastern Washtenaw County, Michigan, USA
0 750 1,500 3,000 4,500 6,000 Meters
Date: 2/14/2017
Author: Jiachen Wei
Coordinate System: WGS 1984 UTM Zone 17N
Projection: Transverse Mercator
Datum: WGS 1984
False Easting: 500,000.0000
False Northing: 0.0000
Central Meridian: -81.0000
Scale Factor: 0.9996
Latitude Of Origin: 0.0000
Units: Meter
nature
agriculture
development
unavailable

With 62.06% of the plannable land being conflicting, we should not neglect the imitations and uncertainties involved in our decision rule and the corresponding algorithm. First off, land-use noise is prevalent in the future land use map, especially in nature areas where the three types of land use strongly compete. This matter is not as simple as filtering out the noise and in-situ research is preferred to investigate how to balance the three criteria in these *noisy* areas. Second, uncertainty is inherent in thresholding. For example, it may not be reasonable to treat a 35-unit and a 34-unit difference as two separate categories. Also intrinsic to our algorithm, when traversing from *array[0,0]* to *array[maxRow,maxCol],* it is likely to allocate to smaller-difference locations (say, dev-ag=35), and miss out the bigger-difference locations (say, dev-ag=49) because the number of dev cells reaches the goal before the more favorable locations in the array are visited. Third, we need to probe for the “perfect” thresholds to achieve the percentage goal and avoid logical mistakes. The exploring process can be inefficient, or even hopeless (as in the case where we put agriculture before nature). Fourth, it is advisable to alternate between “competitors” with comparable thresholds. For example, when allocating development locations, it makes sense to follow another sequence [dev-ag>25, dev-nat>35, dev>ag, dev-nat>15] instead of competing with one and then another. The thresholds can also be standardized through discussion with SEMCOG and review of relevant literature.

IV. Appendix: the Python Script

i. Exported Model

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# suit_Nat.py
# Created on: 2017-02-10 21:00:00.00000
#   (generated by ArcGIS/ModelBuilder)
# Usage: suit_Nat <nedem> <nelu95> <nestream> <nepreset> <suiNa>
# Description: generating nature suitability map
# Creator and Modifier: Jiachen Wei
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy
from arcpy import env
from arcpy.sa import *
arcpy.env.overwriteOutput = True

# Script arguments
nedem = arcpy.GetParameterAsText(0)
if nedem == '#' or not nedem:
    nedem = "N:\\381 A2\\Group 14\\Group 14\\data\\nedem" # provide a default value if unspecified

nelu95 = arcpy.GetParameterAsText(1)
if nelu95 == '#' or not nelu95:
    nelu95 = "N:\\381 A2\\Group 14\\Group 14\\data\\nelu95" # provide a default value if unspecified

nestream = arcpy.GetParameterAsText(2)
```

```
if nestream == '#' or not nestream:
    nestream = "N:\\381 A2\\Group 14\\Group 14\\data\\nestream" # provide a
default value if unspecified

nepreset = arcpy.GetParameterAsText(3)
if nepreset == '#' or not nepreset:
    nepreset = "N:\\381 A2\\Group 14\\Group 14\\data\\nepreset" # provide a default
value if unspecified

suiNa = arcpy.GetParameterAsText(4)
if suiNa == '#' or not suiNa:
    suiNa = "N:\\381 A2\\Group 14\\Group 14\\suiNa" # provide a default value if
unspecified

# Local variables:
#Reclass_nelu3 = "N:\\381 A2\\Group 14\\Group 14\\scratch.gdb\\Reclass_nelu3"
#Output_direction_raster = ""
#Output_direction_raster__2_ = ""
na_Slope_nedem1 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\na_Slope_nedem1"
na_Slice_na_Slo1 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\na_Slice_na_Slo1"
na_EucDist_na_r1 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\EucDist_Recl2"
na_FuzzyMe_na_E1 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\na_FuzzyMe_na_E1"
v100 = "100"
na_Times_na_Fuz1 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Times_na_Fuz1"
v101 = "101"
```

```
na_EucDist_nest1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\na_EucDist_d2streams"
na_Slice_na_Euc1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\na_Slice_na_Euc1"
na_Minus_1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\na_Minus_1"
Reclass_nepr3 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nepr3"
na_Weighte_na_S1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Weighte_na_S2"
Reclass_nelu4 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nelu4"
na_Times_na_Wei1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Times_Weight3"

# Set Geoprocessing environments
# arcpy.env.scratchWorkspace = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb"
# arcpy.env.workspace = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb"

#----------------------distance to wetlands: one criterion in nature suitability evaluation----------------------
# Process: Reclassify 1995 wetland using nelu95 data file
wetland_Reclassify = Reclassify(nelu95, "VALUE", "111 NODATA;112 NODATA;113 NODATA;115 NODATA;121 NODATA;122 NODATA;124 NODATA;126 NODATA;128 NODATA;131 NODATA;134 NODATA;138 NODATA;141 NODATA;142 NODATA;144 NODATA;145 NODATA;146 NODATA;171 NODATA;172 NODATA;193 NODATA;194 NODATA;210 NODATA;220 NODATA;230 NODATA;240 NODATA;250 NODATA;290 NODATA;310 NODATA;320 NODATA;412 NODATA;421 NODATA;429 NODATA;510 NODATA;520 NODATA;530 NODATA;611 1;613 1;622 1;625 1", "NODATA")
# Process: Euclidean Distance to wetland
arcpy.gp.EucDistance_sa(wetland_Reclassify, na_EucDist_na_r1, "", "30", "")
# Process: Fuzzy Membership
```

```
arcpy.gp.FuzzyMembership_sa(na_EucDist_na_r1, na_FuzzyMe_na_E1,
"LINEAR 4000 30", "NONE")
# Process: Times standarizing distance to welands
arcpy.gp.Times_sa(na_FuzzyMe_na_E1, v100, na_Times_na_Fuz1)

#----------------------distance to streams: one criterion in nature suitability evaluation-
---------------------
# Process: Euclidean Distance(2) to streams using nestream data file
arcpy.gp.EucDistance_sa(nestream, na_EucDist_nest1, "", "30", "")
# Process: Slice (2) standarizing distance to streams
arcpy.gp.Slice_sa(na_EucDist_nest1, na_Slice_na_Euc1, "100",
"EQUAL_INTERVAL", "1")
# Process: 101 Minus rater values
arcpy.gp.Minus_sa(v101, na_Slice_na_Euc1, na_Minus_1)

#----------------------slope: one criterion in nature suitability evaluation-------------------
---
# Process: Slope from DEM data
arcpy.gp.Slope_sa(nedem, na_Slope_nedem1, "DEGREE", "1")
# Process: Slice slope data
arcpy.gp.Slice_sa(na_Slope_nedem1, na_Slice_na_Slo1, "100",
"EQUAL_INTERVAL", "1")

#----------------------distance to wetlands: one criterion in nature suitability
evaluation----------------------
# Process: Reclassify (2) presettlement vegetation according to relative
importance
arcpy.gp.Reclassify_sa(nepreset, "VALUE", "3320 75;3360 75;4100 50;4121
50;4122 50;4123 50;4140 50;4200 50;4233 50;5100 100;5200 100;6122 100;6123
100;6221 100;6226 100;6227 100", Reclass_nepr3, "DATA")
```

```
#---------------------summing the criteria and adding availability constraint-------------
---------
# Process: Weighted Sum
arcpy.gp.WeightedSum_sa("'N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\na_Slice_na_Slo1' Value 0.08;'N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Times_na_Fuz1' VALUE 0.64;'N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\na_Minus_1' Value 0.21;'N:\\381 A2\\Group 14\\Group 14\\scratch.gdb\\Reclass_nepr3' Value 0.07", na_Weighte_na_S1)
# Process: Reclassify (3) available land
arcpy.gp.Reclassify_sa(nelu95, "VALUE", "111 0;112 0;113 0;115 0;121 0;122 0;124 0;126 0;128 0;131 0;134 0;138 0;141 0;142 0;144 0;145 0;146 0;171 1;172 1;193 0;194 0;210 1;220 1;230 1;240 1;250 1;290 1;310 1;320 1;412 1;421 1;429 1;510 0;520 0;530 0;611 0;613 0;622 0;625 0", Reclass_nelu4, "DATA")
# Process: Times (2) weighted sum * available land
arcpy.gp.Times_sa(na_Weighte_na_S1, Reclass_nelu4, na_Times_na_Wei1)
# Process: Slice (3) standarizing the Times(2) result to 1-100
suitNa=Slice(na_Times_na_Wei1, "100", "EQUAL_AREA", "1")
suitNa.save("N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiNa")  #delete suitNa in ArcGIS before running again

# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# suit_Ag.py
# Created on: 2017-02-10 14:02:49.00000
#   (generated by ArcGIS/ModelBuilder)
# Usage: suit_Ag <nedem_> <nestream_> <neclayp_> <nelu95_> <suitAg> <AvailableLand>
# Description: generating agriculture suitability map
# Creator: Zhihao Wang
#Modifier: Jiachen Wei
# ---------------------------------------------------------------------------
# Import arcpy module
import arcpy
from arcpy import env
from arcpy.sa import *
arcpy.env.overwriteOutput = True
#arcpy.env.workspace = "N:\\381 A2\\Group 14\\Group 14\\data\\"

# Script arguments
nedem_ = arcpy.GetParameterAsText(0)
if nedem_ == '#' or not nedem_:
    nedem_ = "N:\\381 A2\\Group 14\\Group 14\\data\\nedem" # provide a default
value if unspecified

nestream_ = arcpy.GetParameterAsText(1)
if nestream_ == '#' or not nestream_:
    nestream_ = "N:\\381 A2\\Group 14\\Group 14\\data\\nestream" # provide a
default value if unspecified

neclayp_ = arcpy.GetParameterAsText(2)
if neclayp_ == '#' or not neclayp_:
    neclayp_ = "N:\\381 A2\\Group 14\\Group 14\\data\\neclayp" # provide a default
value if unspecified

nelu95_ = arcpy.GetParameterAsText(3)
if nelu95_ == '#' or not nelu95_:
    nelu95_ = "N:\\381 A2\\Group 14\\Group 14\\data\\nelu95" # provide a default
value if unspecified

suitAg = arcpy.GetParameterAsText(4)
if suitAg == '#' or not suitAg:
    suitAg = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suitAg" # provide a
default value if unspecified

AvailableLand = arcpy.GetParameterAsText(5)
if AvailableLand == '#' or not AvailableLand:
    AvailableLand = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\AvailableLand"
# provide a default value if unspecified

# Local variables:
#Wetland = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_wet"
#Output_direction_raster = ""
Developed_Areas = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_dev"
#Output_direction_raster__2_ = ""
#Output_direction_raster__3_ = ""
v101 = "101"
Scores_clay = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Scores_clay"
Slice_Scores1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_Scores1"
ag_clay = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_clay"
#Slope_ = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slope"
Scores_slope = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Scores_slope"
Slice_Scores3 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_Scores2"
ag_slope = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_slope"
#EucDist_stream = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\EucDist_stream"
FuzzyMe_EucD4 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\FuzzyMe_EucD4"
ag_stream = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_stream"
#EucDist_Wetland = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\EucDist_Wetland"
FuzzyMe_EucD5 = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\FuzzyMe_EucD5"
```

```
ag_wetland = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_wetland"
#EucDist_dev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_dev"
Reclass_EucD1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_EucD1"
FuzzyMe_EucD6 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD6"
ag_develpoed = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_develpoed"
Minus_2 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_2"
Con_Reclass_3 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Con_Reclass_3"
ag_dev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_dev"
Weighte_ag_factors = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Weighte_ag_factors"
ag_landexisting = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\ag_existing"
Times_Weight1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Times_Weight1"
suitAg_nonSlaced = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Times_Times_1"

#----------------------distance to wetlands: one criterion in agriculture suitability evaluation----------------------
# Process: Reclassify (3) 1995 wetland
Wetland = Reclassify(nelu95_, "VALUE", "111 NODATA;112 NODATA;113 NODATA;115 NODATA;121 NODATA;122 NODATA;124 NODATA;126 NODATA;128 NODATA;131 NODATA;134 NODATA;138 NODATA;141 NODATA;142 NODATA;144 NODATA;145 NODATA;146 NODATA;171 NODATA;172 NODATA;193 NODATA;194 NODATA;210 NODATA;220 NODATA;230 NODATA;240 NODATA;250 NODATA;290 NODATA;310 NODATA;320 NODATA;412 NODATA;421 NODATA;429 NODATA;510 NODATA;520 NODATA;530 NODATA;611 1;613 1;622 1;625 1", "NODATA")
# Process: Euclidean Distance to wetland
```

```
EucDist_Wetland= EucDistance(Wetland, "", "30", "")
# Process: Fuzzy Membership (2)
arcpy.gp.FuzzyMembership_sa(EucDist_Wetland, FuzzyMe_EucD5, "LINEAR
300 1500", "SOMEWHAT")
# Process: Slice (2)
arcpy.gp.Slice_sa(FuzzyMe_EucD5, ag_wetland, "100", "EQUAL_INTERVAL",
"1")

#----------------------distance to development areas: one criterion in agriculture
suitability evaluation----------------------
# Process: Reclassify (2) develpment area
dev_Reclassify=Reclassify(nelu95_, "VALUE", "111 1;112 1;113 1;115 1;121
1;122 1;124 1;126 1;128 1;131 1;134 1;138 1;141 1;142 1;144 1;145 1;146 1;171
0;172 0;193 1;194 1;210 NODATA;220 NODATA;230 NODATA;240 NODATA;250
NODATA;290 NODATA;310 NODATA;320 NODATA;412 NODATA;421
NODATA;429 NODATA;510 NODATA;520 NODATA;530 NODATA;611
NODATA;613 NODATA;622 NODATA;625 NODATA", "DATA")
# Process: Euclidean Distance (2) to develpment area
EucDist_dev=EucDistance(dev_Reclassify, "", "30", "")
# Process: Reclassify (6) distance to development areas
arcpy.gp.Reclassify_sa(EucDist_dev, "VALUE", "0 500 0;500 800 1;800 1200
2;1200 1500 3;1500 2000 4;2000 2020.717651 5", Reclass_EucD1, "DATA")
# Process: Fuzzy Membership (3) distance to development areas
arcpy.gp.FuzzyMembership_sa(EucDist_dev, FuzzyMe_EucD6, "LINEAR 0
2020.71765136719", "SOMEWHAT")
# Process: Slice (3) standarizing distance to development areas
arcpy.gp.Slice_sa(FuzzyMe_EucD6, ag_develpoed, "100", "EQUAL_INTERVAL",
"1")
# Process: Minus (3) 101-standarized distance to development areas
arcpy.gp.Minus_sa(v101, ag_develpoed, Minus_2)
# Process: Con to conditionally combine distance to development areas
```

```
arcpy.gp.Con_sa(Reclass_EucD1, ag_develpoed, Con_Reclass_3, Minus_2,
"VALUE < 4")
# Process: Slice (7) Con result
arcpy.gp.Slice_sa(Con_Reclass_3, ag_dev, "100", "EQUAL_INTERVAL", "1")

#----------------------distance to streams: one criterion in agriculture suitability
evaluation----------------------
# Process: Euclidean Distance (3) from streams data
EucDist_stream =EucDistance(nestream_, "", "30", "")
# Process: Fuzzy Membership distance to streams
arcpy.gp.FuzzyMembership_sa(EucDist_stream, FuzzyMe_EucD4, "LINEAR 300
1500", "NONE")
# Process: Slice (4) distance to streams
arcpy.Slice_3d(FuzzyMe_EucD4, ag_stream, "100", "EQUAL_INTERVAL", "1")

#----------------------clay content: one criterion in agriculture suitability evaluation----
------------------
# Process: Reclassify (8) clay content
arcpy.gp.Reclassify_sa(neclayp_, "VALUE", "9 1;10 1;11 1;15 2;24 3;30 3",
Scores_clay, "DATA")
# Process: Slice (5) Reclassify (8) result
arcpy.gp.Slice_sa(Scores_clay, Slice_Scores1, "100", "EQUAL_INTERVAL", "1")
# Process: Minus (2) 101-clay
arcpy.gp.Minus_sa(v101, Slice_Scores1, ag_clay)

#----------------------slope: one criterion in agriculture suitability evaluation-------------
---------
# Process: Slope
na_Slope_nedem1=Slope(nedem_, "DEGREE", "1")
# Process: Reclassify (4) slope
```

```
Scores_slope=Reclassify(na_Slope_nedem1, "VALUE", "0 3 1;3 5 2;5 8 3;8 15 4",
"DATA")
# Process: Slice slope
arcpy.gp.Slice_sa(Scores_slope, Slice_Scores3, "100", "EQUAL_INTERVAL",
"1")
# Process: Minus 101-standarized slope
arcpy.gp.Minus_sa(v101, Slice_Scores3, ag_slope)

#----------------------summing the criteria, and adding agricultue and availability
constraint----------------------
# Process: Weighted Sum
arcpy.gp.WeightedSum_sa("'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\ag_clay' Value 0.18;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\ag_slope' Value 0.44;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\ag_stream' Value 0.11;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\ag_wetland' Value 0.11;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\ag_dev' VALUE 0.15", Weighte_ag_factors)
# Process: Reclassify 1995 agricultual land as a constraint
arcpy.gp.Reclassify_sa(nelu95_, "VALUE", "111 0;112 0;113 0;115 0;121 0;122
0;124 0;126 0;128 0;131 0;134 0;138 0;141 0;142 0;144 0;145 0;146 0;171 0;172
0;193 0;194 0;210 1;220 1;230 1;240 1;250 1;290 1;310 0;320 0;412 0;421 0;429
0;510 0;520 0;530 0;611 0;613 0;622 0;625 0", ag_landexisting, "DATA")
# Process: Times(1) Weighted Sum * 1995 agricultual land
arcpy.gp.Times_sa(Weighte_ag_factors, ag_landexisting, Times_Weight1)
# Process: Reclassify (5) available land
arcpy.gp.Reclassify_sa(nelu95_, "VALUE", "111 0;112 0;113 0;115 0;121 0;122
0;124 0;126 0;128 0;131 0;134 0;138 0;141 0;142 0;144 0;145 0;146 0;171 1;172
1;193 0;194 0;210 1;220 1;230 1;240 1;250 1;290 1;310 1;320 1;412 1;421 1;429
1;510 0;520 0;530 0;611 0;613 0;622 0;625 0", AvailableLand, "DATA")
# Process: Times (2) available land * Times(1) result
arcpy.gp.Times_sa(Times_Weight1, AvailableLand, suitAg_nonSlaced)
```

```
        # Process: Slice (6) result of Times(2)
        suitAg=Slice(suitAg_nonSlaced, "100", "EQUAL_AREA", "1")
suitAg.save("N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiAg")


        # -*- coding: utf-8 -*-
        # ---------------------------------------------------------------------------
        # suit_Dev.py
        # Created on: 2017-02-10 14:01:13.00000
        #   (generated by ArcGIS/ModelBuilder)
        # Usage: suit_Dev <nepublic> <suitDEV> <nedem> <nestream> <nelu95>
        <neaa> <neroads>
        # Description: generating development suitability map
        # Creator: Junzheng Zhang
        #Modifier: Jiachen Wei
        # ---------------------------------------------------------------------------

        # Import arcpy module
        import arcpy
        from arcpy import env
        from arcpy.sa import *
        arcpy.env.overwriteOutput = True

        # Script arguments
        nepublic = arcpy.GetParameterAsText(0)
        if nepublic == '#' or not nepublic:
            nepublic = "N:\\381 A2\\Group 14\\Group 14\\data\\nepublic" # provide a default
        value if unspecified

        suitDEV = arcpy.GetParameterAsText(1)
        if suitDEV == '#' or not suitDEV:
```

```
    suitDEV = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev" # provide a
default value if unspecified

nedem = arcpy.GetParameterAsText(2)
if nedem == '#' or not nedem:
    nedem = "N:\\381 A2\\Group 14\\Group 14\\data\\nedem" # provide a default
value if unspecified

nestream = arcpy.GetParameterAsText(3)
if nestream == '#' or not nestream:
    nestream = "N:\\381 A2\\Group 14\\Group 14\\data\\nestream" # provide a
default value if unspecified

nelu95 = arcpy.GetParameterAsText(4)
if nelu95 == '#' or not nelu95:
    nelu95 = "N:\\381 A2\\Group 14\\Group 14\\data\\nelu95" # provide a default
value if unspecified

neaa = arcpy.GetParameterAsText(5)
if neaa == '#' or not neaa:
    neaa = "N:\\381 A2\\Group 14\\Group 14\\data\\neaa" # provide a default value
if unspecified

neroads = arcpy.GetParameterAsText(6)
if neroads == '#' or not neroads:
    neroads = "N:\\381 A2\\Group 14\\Group 14\\data\\neroads" # provide a default
value if unspecified

# Local variables:
###Output_distance_raster = ""
Reclass_aa = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_neaa1"
```

```
#Output_distance_raster__2_ = ""
Reclass_pub = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nepu1"
##Output_distance_raster__3_ = ""
Reclass_nero1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nero1"
#Output_distance_raster__4_ = ""
Reclass_dev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nelu1"
#Output_distance_raster__5_ = ""
Reclass_wetland = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Reclass_nelu2"
#Output_distance_raster__6_ = ""
v101 = "101"
dev_1 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slope_nedem1"
slopescore = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_Slop1"
dev_15 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_Reclas1"
Minus_slope = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_1"
EucDist_stream = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_nest1"
dev_13 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD1"
stream = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM1"
v101_ = "101"
EucDist_dev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_Recl1"
dev_18 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD2"
dev_19 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM2"
Minus_dev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_3"
EucDist_wetland = "N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\EucDist_Recl2"
dev_21 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD3"
wetland = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM3"
v101__ = "101"
EucDist_aa = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_Recl3"
dev_23 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD7"
dev_24 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM4"
```

```
Minus_aa = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_4"
v101___ = "101"
EucDist_pub = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_Recl4"
dev_25 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD8"
dev_26 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM5"
Minus_public = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_5"
v101____ = "101"
EucDist_road = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\EucDist_Recl5"
dev_28 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\FuzzyMe_EucD9"
dev_29 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Slice_FuzzyM6"
Minus_road = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Minus_6"
dev_31 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Weighte_Minu1"
dev_32 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Reclass_nelu3"
dev_33 = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\Times_Weight2"

#----------------------distance to streams: one criterion in development suitability
evaluation----------------------
# Process: Euclidean Distance to streams
arcpy.gp.EucDistance_sa(nestream, EucDist_stream, "", "30", "")
# Process: Fuzzy Membership for distance to streams
arcpy.gp.FuzzyMembership_sa(EucDist_stream, dev_13, "LINEAR 100 500",
"SOMEWHAT")
# Process: Slice standardizing distance to streams
arcpy.gp.Slice_sa(dev_13, stream, "100", "EQUAL_INTERVAL", "1")

#----------------------distance to city: one criterion in development suitability
evaluation----------------------
# Process: Reclassify (3) urban area
arcpy.gp.Reclassify_sa(neaa, "VALUE", "0 0.500000 NODATA;0.500000 1 1",
Reclass_aa, "DATA")
# Process: Euclidean Distance (2) to Ann Arbor
```

```
arcpy.gp.EucDistance_sa(Reclass_aa, EucDist_aa, "", "30", "")
# Process: Fuzzy Membership (4) for distance to Ann Arbor
arcpy.gp.FuzzyMembership_sa(EucDist_aa, dev_23, "LINEAR 500 2000", "SOMEWHAT")
# Process: Slice(5) Fuzzy Membership (4) result
arcpy.gp.Slice_sa(dev_23, dev_24, "100", "EQUAL_INTERVAL", "1")
# Process: Minus(3) 101-Slice(5) result
arcpy.gp.Minus_sa(v101__, dev_24, Minus_aa)

#----------------------distance to public land: one criterion in development suitability evaluation----------------------
# Process: Reclassify (4) public land
arcpy.gp.Reclassify_sa(nepublic, "VALUE", "0 0.500000 NODATA;0.500000 1 1", Reclass_pub, "DATA")
# Process: Euclidean Distance  (3) to public land
arcpy.gp.EucDistance_sa(Reclass_pub, EucDist_pub, "", "30", "")
# Process: Fuzzy Membership (5) for distance to public land
arcpy.gp.FuzzyMembership_sa(EucDist_pub, dev_25, "LINEAR 300 1500", "SOMEWHAT")
# Process: Slice (6) Fuzzy Membership (5) result
arcpy.gp.Slice_sa(dev_25, dev_26, "100", "EQUAL_INTERVAL", "1")
# Process: Minus (4) 101- Slice (6)result
arcpy.gp.Minus_sa(v101___, dev_26, Minus_public)

#----------------------distance to major roads: one criterion in development suitability evaluation----------------------
# Process: Reclassify (5) major roads
arcpy.gp.Reclassify_sa(neroads, "VALUE", "2 NODATA;3 1;4 NODATA;9 NODATA;18 1", Reclass_nero1, "DATA")
# Process: Euclidean Distance (4) to major roads
arcpy.gp.EucDistance_sa(Reclass_nero1, EucDist_road, "", "30", "")
```

```
# Process: Fuzzy Membership (6) for distance to major roads
arcpy.gp.FuzzyMembership_sa(EucDist_road, dev_28, "LINEAR 300 1500",
"SOMEWHAT")
# Process: Slice (7) Fuzzy Membership (6) result
arcpy.gp.Slice_sa(dev_28, dev_29, "100", "EQUAL_INTERVAL", "1")
# Process: Minus (5) 101 - Slice (7)  result
arcpy.gp.Minus_sa(v101____, dev_29, Minus_road)

#----------------------distance to development land: one criterion in development
suitability evaluation----------------------
# Process: Reclassify development land
arcpy.gp.Reclassify_sa(nelu95, "VALUE", "111 170 1;170 180 NODATA;180 200
1;200 625 NODATA", Reclass_dev, "DATA")
# Process: Euclidean Distance (5) to development land
arcpy.gp.EucDistance_sa(Reclass_dev, EucDist_dev, "", "30", "")
# Process: Fuzzy Membership (2) for distance to development land
arcpy.gp.FuzzyMembership_sa(EucDist_dev, dev_18, "LINEAR 500 2000",
"SOMEWHAT")
# Process: Slice (3) standardizing distance to distance to development land
arcpy.gp.Slice_sa(dev_18, dev_19, "100", "EQUAL_INTERVAL", "1")
# Process: Minus (2) 101-slice(3) result
arcpy.gp.Minus_sa(v101_, dev_19, Minus_dev)

#----------------------distance to wetland: one criterion in development suitability
evaluation----------------------
# Process: Reclassify (2) wetland 1995
arcpy.gp.Reclassify_sa(nelu95, "VALUE", "111 600 NODATA;600 625 1",
Reclass_wetland, "DATA")
# Process: Euclidean Distance(6) to wetland 1995
arcpy.gp.EucDistance_sa(Reclass_wetland, EucDist_wetland, "", "30", "")
# Process: Fuzzy Membership (3) for distance to wetland 1995
```

```
arcpy.gp.FuzzyMembership_sa(EucDist_wetland, dev_21, "LINEAR 300 1000",
"SOMEWHAT")
# Process: Slice (4) Fuzzy Membership(3) result
arcpy.gp.Slice_sa(dev_21, wetland, "100", "EQUAL_INTERVAL", "1")

#---------------------slope: one criterion in development suitability evaluation----------
------------
# Process: Slope from dem data
arcpy.gp.Slope_sa(nedem, dev_1, "DEGREE", "1")
# Process: Reclassify (6) slope
arcpy.gp.Reclassify_sa(dev_1, "Value", "0 0.300000 2;0.300000 2 1;2 5 2;5 10
3;10 25 4;25 70.671387 5", slopescore, "DATA")
# Process: Slice(2) standardizing slope
arcpy.gp.Slice_sa(slopescore, dev_15, "100", "EQUAL_INTERVAL", "1")
# Process: Minus 101-slope scores
arcpy.gp.Minus_sa(v101, dev_15, Minus_slope)

#---------------------summing the criteria and adding availability constraint-------------
---------
# Process: Weighted Sum
arcpy.gp.WeightedSum_sa("'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Minus_1' Value 0.028;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Slice_FuzzyM1' Value 0.065;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Minus_3' Value 0.129;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Slice_FuzzyM3' Value 0.049;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Minus_4' Value 0.186;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Minus_5' Value 0.336;'N:\\381 A2\\Group 14\\Group
14\\Scratch.gdb\\Minus_6' Value 0.207", dev_31)
# Process: Reclassify (7) available land
arcpy.gp.Reclassify_sa(nelu95, "VALUE", "111 150 0;150 180 1;180 200 0;200
450 1;450 625 0", dev_32, "DATA")
```

```
# Process: Times Weighted Sum*Reclassify (7)  result
arcpy.gp.Times_sa(dev_31, dev_32, dev_33)
# Process: Slice (8) standarizing Times result
suitDev=Slice(dev_33, "100", "EQUAL_AREA", "1")
suitDev.save("N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev")
```

ii. The Conflict Map and the Future Land Use Map

```
# ---------------------------------------------------------------------------
# conflictMap.py
# Created on: 2017-02-10 19:45:15.00000
# Acknowledgements: suit_Ag created by Zhihao Wang, suit_Dev created by Junzheng Zhang, and instuctions fot Assignment 1b and 2 from Derek T. Robinson
# Purpose: to generate conflict map
# Usage: suiAg,suiDev,suiNa
# Description: conflicting locations are where more than one suitability maps reach >50 at this location
# Creator: Jiachen Wei
# Preconditions: input 3 raster maps where suitability values are integers from 1 to 100, with 1 accounting for unavailable/unplannable lands
# Postconditions:
#         conflict map from cflCount[][] where unavailable=0, dev=1, ag=3, nat=5, dev+ag=4, dev+nat=6, ag+nat=8, all conflicting=9, available and none>50 =10
#         future land use map from fluArray where unavailable=0, dev=1, ag=2, nat=3 (devGoal cells, agGoal cells, remaining cells)
# ---------------------------------------------------------------------------
# Import library and set envrionment
import time
import numpy
import arcpy
from arcpy import env
from arcpy.sa import *
arcpy.env.overwriteOutput = True
arcpy.env.cellSize = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev"
arcpy.env.snapRaster = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev"
arcpy.env.extent = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev"
startTime=time.time()

# string object containing address for reaster files
suiDev = arcpy.GetParameterAsText(0)
if suiDev == '#' or not suiDev:
   suiDev = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiDev" # provide a
default value if unspecified

suiAg = arcpy.GetParameterAsText(1)
if suiAg == '#' or not suiAg:
   suiAg = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiAg" # provide a
default value if unspecified

suiNa = arcpy.GetParameterAsText(2)
if suiNa == '#' or not suiNa:
   suiNa = "N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\suiNa" # provide a
default value if unspecified

#----------------------read raster and convert to numpy arrays----------------------
raster2arrayStartTime=time.time()
# RasterToNumPyArray (in_raster, {lower_left_corner}, {ncols}, {nrows},
{nodata_to_value})
devRaster= arcpy.Raster(suiDev)
lowerLeft = arcpy.Point(devRaster.extent.XMin, devRaster.extent.YMin)
```

```
devArray=arcpy.RasterToNumPyArray(devRaster,lowerLeft,"","","")

agRaster= arcpy.Raster(suiAg)
lowerLeft = arcpy.Point(agRaster.extent.XMin, agRaster.extent.YMin)
agArray=arcpy.RasterToNumPyArray(agRaster,lowerLeft,"","","")

natRaster= arcpy.Raster(suiNa)
lowerLeft = arcpy.Point(natRaster.extent.XMin, natRaster.extent.YMin)
natArray=arcpy.RasterToNumPyArray(natRaster,lowerLeft,"","","")

print "read and convert: "+ str(time.time()-raster2arrayStartTime)+"seconds"

#----------------------create the conflict map----------------------
cflMapStartTime=time.time()

#addressing difference in row and column numbers of the three arrays
devRows=devArray.shape[0]       #608
devCols=devArray.shape[1]       #702
print "devRows =" +str(devRows) + " devCols =" +str(devCols)
agRows=agArray.shape[0]         #607
agCols=agArray.shape[1]         #710
print "agRows =" +str(agRows) + " agCols =" +str(agCols)
natRows=natArray.shape[0]       #607
natCols=natArray.shape[1] #710
print "natRows =" +str(natRows) + " natCols =" +str(natCols)
numRows=min(devRows,agRows,natRows) #minimum number of cells
numCols=min(devCols,agCols,natCols)

#generating the conflict array (the mismatching rows and columns are deleted)
cflCount=numpy.zeros((numRows, numCols)) #a numpy array of 0
```

```
#chunk statements by proper indentation
#range() from 0 to row-1
for row in range(numRows):
 for col in range(numCols):
            if devArray[row,col]>50:
             cflCount[row,col]= cflCount[row,col]+1
                     #devCount= devCount+1
                     #print str(devArray[row,col])+" "+str(cflCount[row,col])
for row in range(numRows):
           for col in range(numCols):
            if agArray[row,col]>50: cflCount[row,col]= cflCount[row,col]+3
for row in range(numRows):
           for col in range(numCols):
            if natArray[row,col]>50: cflCount[row,col]= cflCount[row,col]+5
for row in range(numRows):
           for col in range(numCols):
            if devArray[row,col]>1 and devArray[row,col]<51 and agArray[row,col]>1
and agArray[row,col]<51 and natArray[row,col]>1 and natArray[row,col]<51:
cflCount[row,col]=10

#print str(devCount)



#NumPyArrayToRaster (in_array, {lower_left_corner}, {x_cell_size}, {y_cell_size},
{value_to_nodata})
cflRaster=arcpy.NumPyArrayToRaster(cflCount,lowerLeft,devRaster.meanCellWi
dth,devRaster.meanCellHeight,"")  #same cell size as input rasters
arcpy.DefineProjection_management(cflRaster,devRaster.spatialReference)
cflRaster.save("N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\cflMap")
print "conflict map: "+str(time.time()-cflMapStartTime)+"seconds"
```

```
#----------------------create the future land use map----------------------
#future land use array and count of the 3 land use types
fluArray=numpy.zeros((numRows, numCols))
dev=0
ag=0
nat=0
devGoal=28905      #available land *10%
agGoal=104059      #available land *36%
natGoal=156088     #available land *54%

#allocating development land (cflCount 1,4,6,9,10)----------------------
allocateDevStartTime=time.time()

#no conflict with development: cflCount[row,col]=1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==1:
            fluArray[row,col]=1
            dev=dev+1

#dev conflicting with ag: cflCount[row,col]=4 (difference thresholds: >35, >15, >0)
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==4 and devArray[row,col]-agArray[row,col] >=35:
            fluArray[row,col]=1
            dev=dev+1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==4 and devArray[row,col]-agArray[row,col] <35 and
devArray[row,col]-agArray[row,col] >=15:
            fluArray[row,col]=1
            dev=dev+1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==4 and devArray[row,col]-agArray[row,col] <15 and
devArray[row,col]>agArray[row,col] :
            fluArray[row,col]=1
            dev=dev+1

#dev conflicting with nat: cflCount[row,col]=6 (difference thresholds: >35, >15;
cannot be lower for preservation purposes)
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==6 and devArray[row,col]-natArray[row,col] >=35:
            fluArray[row,col]=1
            dev=dev+1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==6 and devArray[row,col]-natArray[row,col] <35 and
devArray[row,col]-natArray[row,col]>=15:
            fluArray[row,col]=1
            dev=dev+1

#dev conflicting with both ag and nat: cflCount[row,col][row,col]=9 (difference
thresholds: 5>=both, then 10>=either one; cannot be lower for preservation
purposes)
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==9 and devArray[row,col]-agArray[row,col] >=5 and
devArray[row,col]-natArray[row,col] >=5:
            fluArray[row,col]=1
            dev=dev+1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==9 and devArray[row,col]-agArray[row,col] >=10 or devArray[row,col]-natArray[row,col] >=10:
            fluArray[row,col]=1
            dev=dev+1

#dev, ag, nat all below 50 but allocate to dev: cflCount[row,col][row,col]=10 (difference thresholds: 5>=both, then 10>=either one; cannot be lower for preservation purposes)
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==10 and devArray[row,col]-agArray[row,col] >=5 and devArray[row,col]-natArray[row,col] >=5:
            fluArray[row,col]=1
            dev=dev+1
for row in range(numRows):
          for col in range(numCols):
           if dev>=devGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif cflCount[row,col] ==10 and devArray[row,col]-agArray[row,col] >=10 or devArray[row,col]-natArray[row,col] >=10:
            fluArray[row,col]=1
            dev=dev+1

print "allocate development land use: "+str(time.time()-
allocateDevStartTime)+"seconds"



#allocating nature land (cflCount 5,6,8,9,10 )----------------------
allocateNatStartTime=time.time()

#no conflict with nature: cflCount[row,col]=5
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==5:
            fluArray[row,col]=3
            nat=nat+1

# nat conflicting with dev: cflCount[row,col][row,col]=6 (now that devGoal is fulfilled,
difference thresholds: >=25, >0)
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
```

```
           elif fluArray[row,col]==0 and cflCount[row,col] ==6 and natArray[row,col]-
devArray[row,col] >=25:
            fluArray[row,col]=3
            nat=nat+1
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==6 and natArray[row,col]-
devArray[row,col] <25 and natArray[row,col]-devArray[row,col] >=0:
            fluArray[row,col]=3
            nat=nat+1

# nat conflicting with ag: cflCount[row,col][row,col]=8 (difference thresholds: >=35,
>=5; cannot be lower for preservation purposes)
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==8 and natArray[row,col]-
agArray[row,col] >=35:
            fluArray[row,col]=3
            nat=nat+1
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==8 and natArray[row,col]-agArray[row,col] <35 and natArray[row,col]-agArray[row,col]>=5:
            fluArray[row,col]=3
            nat=nat+1

#nat conflicting with both dev and ag: cflCount[row,col][row,col]=9 (difference thresholds: >dev and 5>=ag; cannot be lower for preservation purposes)
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==9 and agArray[row,col] >=5 and natArray[row,col]> devArray[row,col]:
            fluArray[row,col]=3
            nat=nat+1

#dev, ag, nat all below 50 but allocate to nat: cflCount[row,col][row,col]=10 (difference thresholds: >dev and >ag)
for row in range(numRows):
          for col in range(numCols):
           if nat>=natGoal:
            break
            break  #break the nested for-loops
           if cflCount[row,col] ==0: continue #unavailable land
           elif fluArray[row,col]==0 and cflCount[row,col] ==10 and natArray[row,col]-agArray[row,col] >=0 and natArray[row,col]- devArray[row,col]>=0:
            fluArray[row,col]=3
            nat=nat+1

    print "allocate nature land use: "+str(time.time()-allocateNatStartTime)+"seconds"

    #allocating agriculture land----------------------
    for row in range(numRows):
            for col in range(numCols):
                    if  fluArray[row,col] ==0 and cflCount[row,col]!=0:
                            fluArray[row,col]=2
                            ag=ag+1

    #report land use proportions and save as raster
    print "dev: " + str(dev)+ ", ag: " +str(ag) + ", nat: "+str(nat)
    fluRaster=arcpy.NumPyArrayToRaster(fluArray,lowerLeft,devRaster.meanCellWi
    dth,devRaster.meanCellHeight,"")
    arcpy.DefineProjection_management(fluRaster,devRaster.spatialReference)
fluRaster.save("N:\\381 A2\\Group 14\\Group 14\\Scratch.gdb\\fluMap")
```

References

ArcMap. (2016). RasterToNumPyArray. Retrieved from https://desktop.arcgis.com/en/arcmap/10.3/analyze/arcpy-functions/rastertonumpyarray-function.htm

Carton, L. J. (2007). *Map making and map use in a multi-actor context: Spatial visualizations and frame conflicts in regional policymaking in the Netherlands* (Doctoral dissertation, TU Delft, Delft University of Technology).

Kyem, P. A. K. (2002). Promoting local community participation in forest management through a PPGIS application in Southern Ghana. *Community participation and geographic information systems*, 218-231.

O'Sullivan, D., & Unwin, D. (2010). Geographic Information Analysis.