ARCPY ALGORITHMS FOR STRATEGIC FARMLAND PRESERVATION

GUY DUER
GEOSPATIAL SOFTWARE DESIGN
PROF. DANA TOMLIN
FALL 2017

Background: Goals of Farmland Preservation

One of the main goals of farmland preservation efforts is the creation of large contiguous farming landscapes. This is discussed thoroughly in the paper “Preserving large farming landscapes: The case of Lancaster County, Pennsylvania” written by Tom Daniels and Lauren Payne-Riley. The information provided in this paper is the motivation for the following report and ArcPy tool.

The literature suggests that farmland preservation is most successful when contiguous areas of at least 250 acres, and ideally 500 acres, are created. Since preservation easements are granted on a parcel by parcel basis, this requires policy-makers and preservationists to focus preservation efforts on specific parcels which would (1) create the largest possible new preserved area, and (2) are strategically located to allow access to other large parcels and connect disparate preserved farm areas. These two considerations contribute to the “attractiveness” of each parcel for preservation. However, as I will show in this report, the large number of parcels and consideration criteria makes identifying the most attractive parcels difficult.

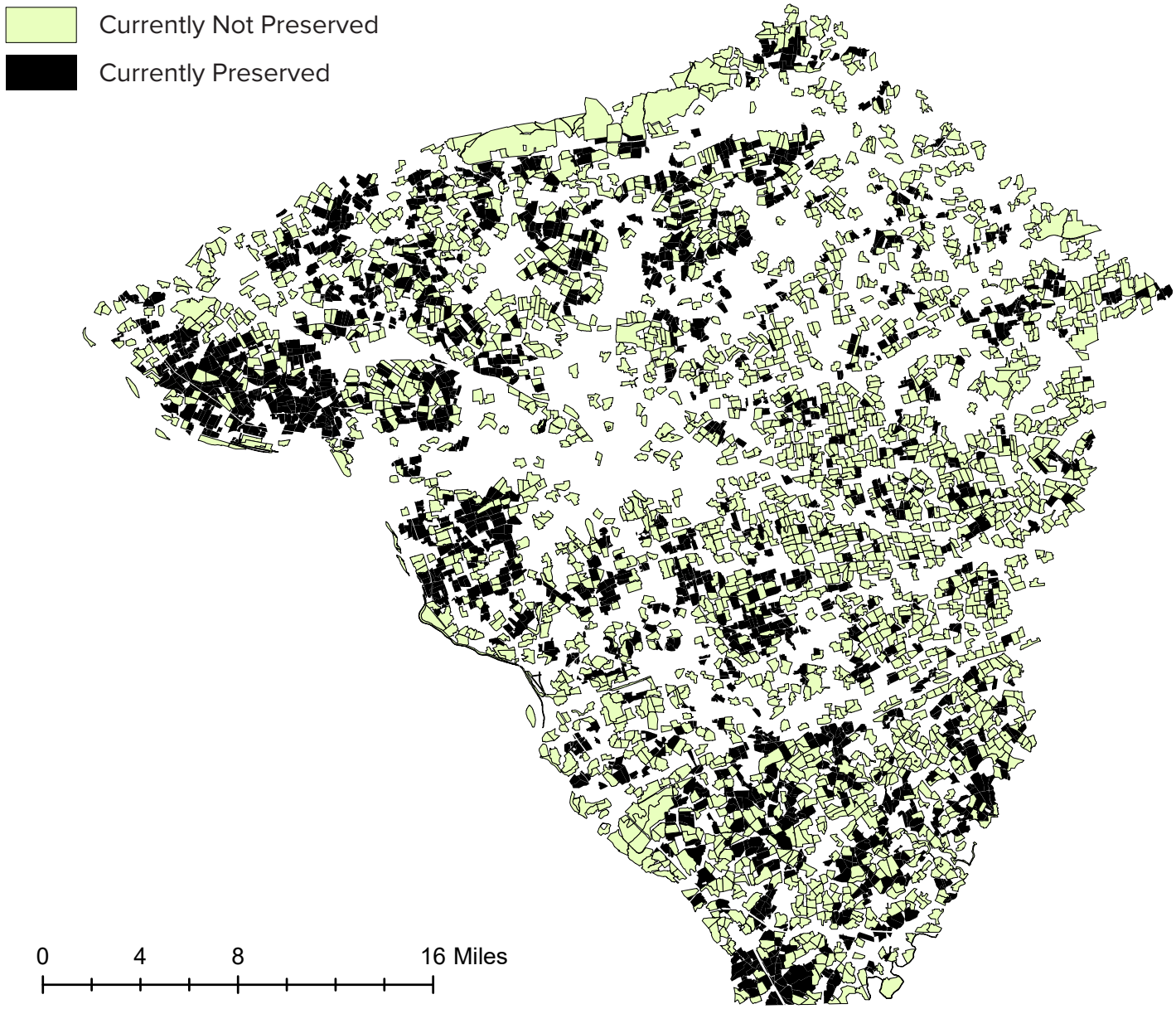
To help in identifying those parcels, I created the “BlobGrower” ArcPy tool. This tool contains two algorithms which work to rank parcels based on various criteria (that will be discussed in this report) as most ideal for preservation. The aim of this tool is to help achieve the most optimal preservation landscape outcomes in the long run by ensuring that all contiguous areas of preserved land are larger than 250 acres.

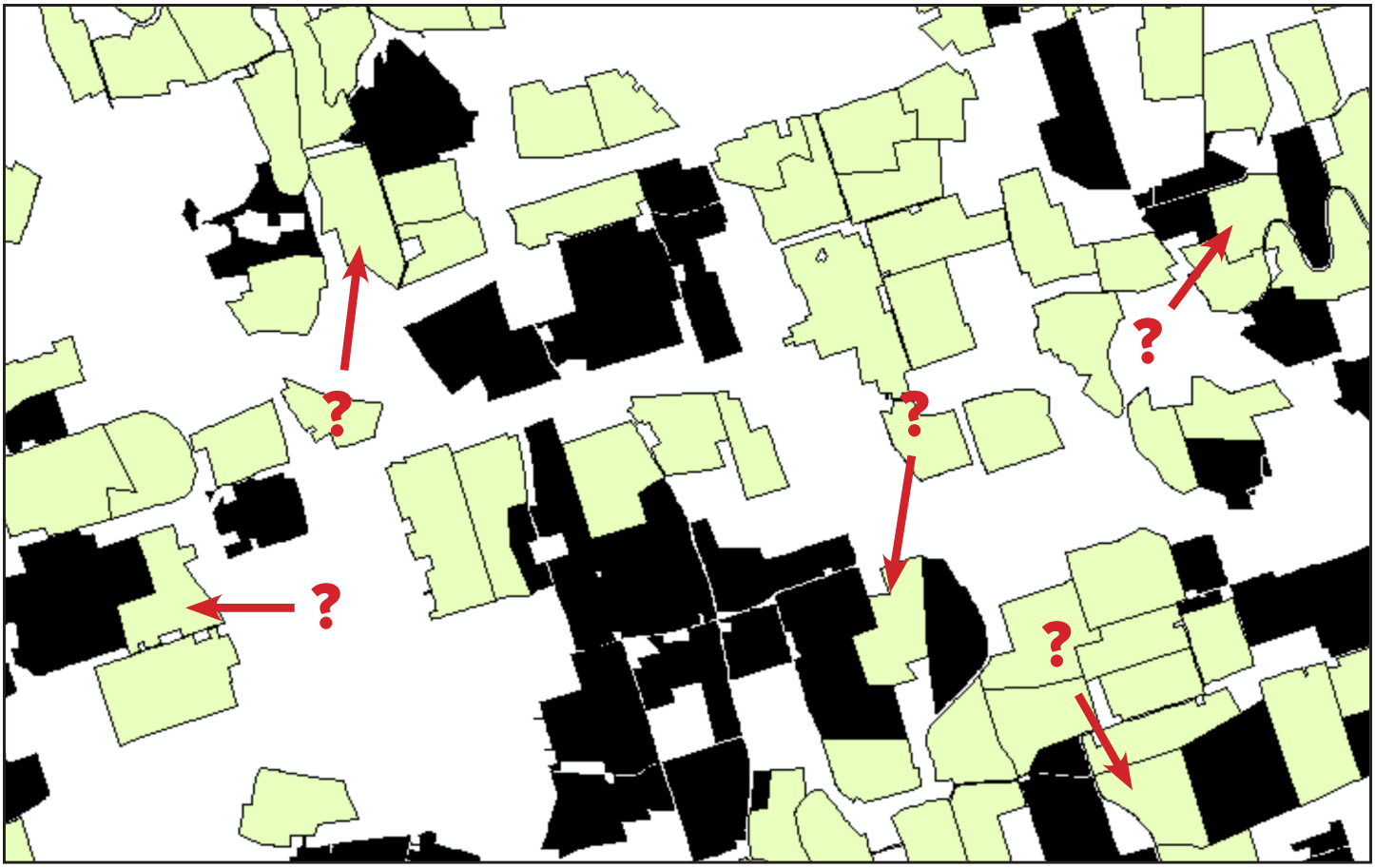
For the purposes of brevity and clarity, I will refer to ‘contiguous areas of preserved land’ as “blobs” for the rest of the report. Hence, the tool’s name- BlobGrower.

Data Set: Farm Parcels in Lancaster County, PA

The dataset that I will use in this analysis contains the shapes and preservation status of all farmland parcels in Lancaster County, Pennsylvania. Typically, parcels smaller than 50 acres are not considered good candidates for preservation due to their small size. Therefore, these parcels have been removed from the dataset.

I will use the BlobGrower tool on this data to identify which currently unpreserved parcels should be prioritized for preservation.





Looking closer to at the parcels reveals the complexity of the problem at hand. With many possible parcels to pursue, preservationists face the following question-

Which parcels, if preserved, will best maximize the number of blobs that are at least 250-500 acres in size, and is most likely to optimize the size and contiguousness of blobs in the long-run?

Greedy vs. Patient Algorithms

Two separate approaches will be used to try to answer this question- a greedy (or naive) method and a patient method. These methods differ in their approach for identifying the most attractive parcels for preservation.

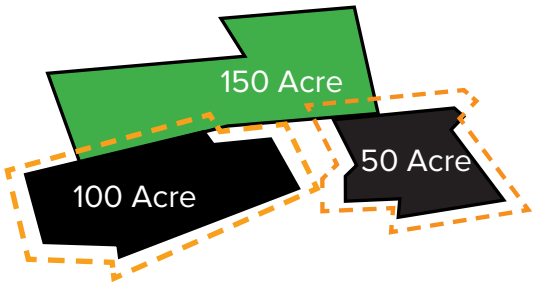
The Greedy Method:

The greedy algorithm takes into account only the current configuration of parcels in its recommendations. The algorithm ranks parcels based on two factors: their own size and

the size of the potential blob that would be created if they were preserved. This algorithm can be defined as follows:

Greedy Score = **Parcel Size** + (**Size of All Adjacent Blobs** + **Parcel Size**)

Example:



= 450 Greedy Score

The Patient Method:

The patient algorithm is more complex, and considers more factors than just the parcel size and immediate potential blob size. Specifically, each parcel is also scored based on the overall land area to which it has access (the total size of adjacent parcels), and the average size of its adjacent parcels. Additionally, each parcel is also evaluated based on the scores of its neighboring parcels, so parcels that are next to very "attractive" parcels also become more attractive. This is done by averaging iterations with the aim of identifying parcels that have the highest potential for unlocking the most optimal outcomes in the long-term.

Since the patient algorithm includes many considerations, the BlobGrower tool allows the user to specify the amount by which each consideration affects the final recommendations.

The patient algorithm can be defined as follows (weights are user defined numbers):

Patient Score = (**Greedy Score** * **Weight1**) + (**Total Neighbor Size** * **Weight2**) + (**Average Neighbor Size** * **Weight3**)

Example:

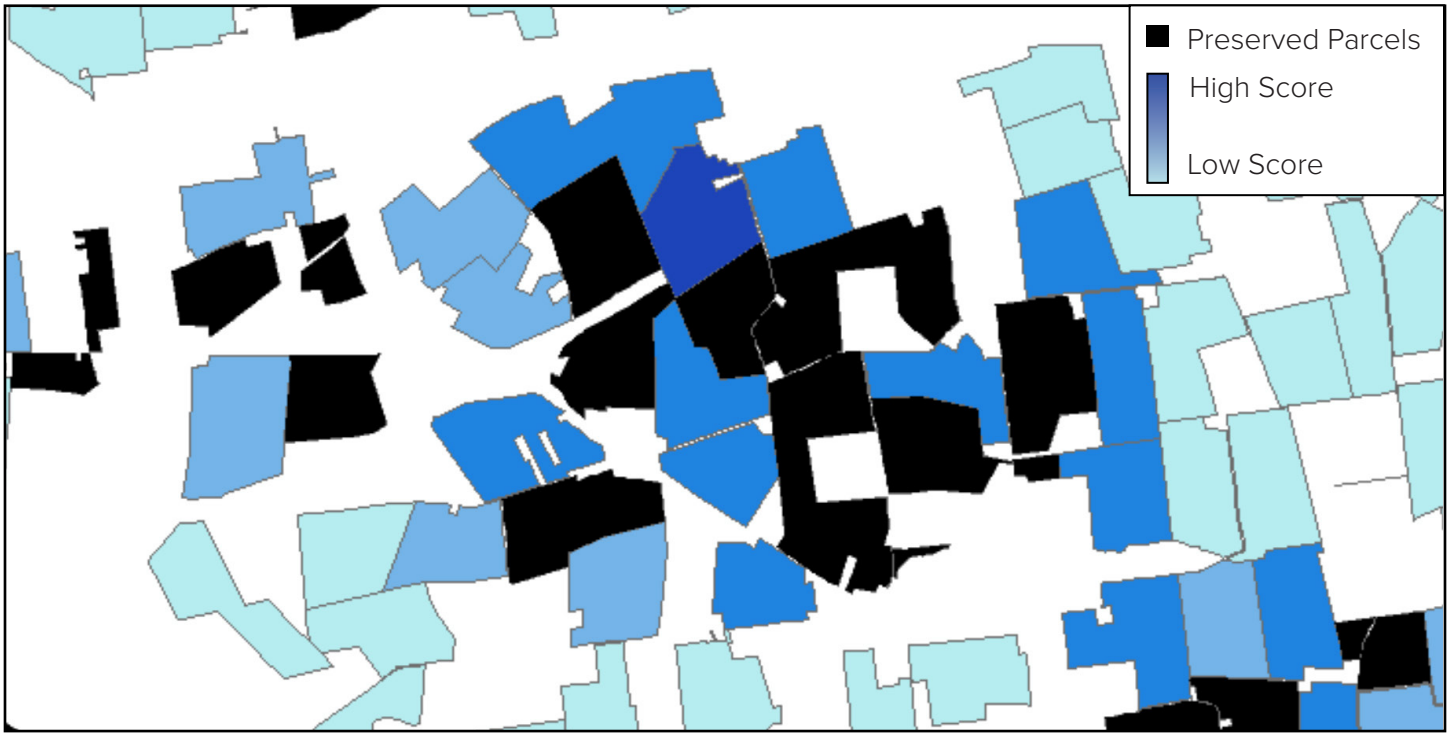
Weight1 = 2
weight2 = 1
weight3 = 4

= 1,530 Patient Score

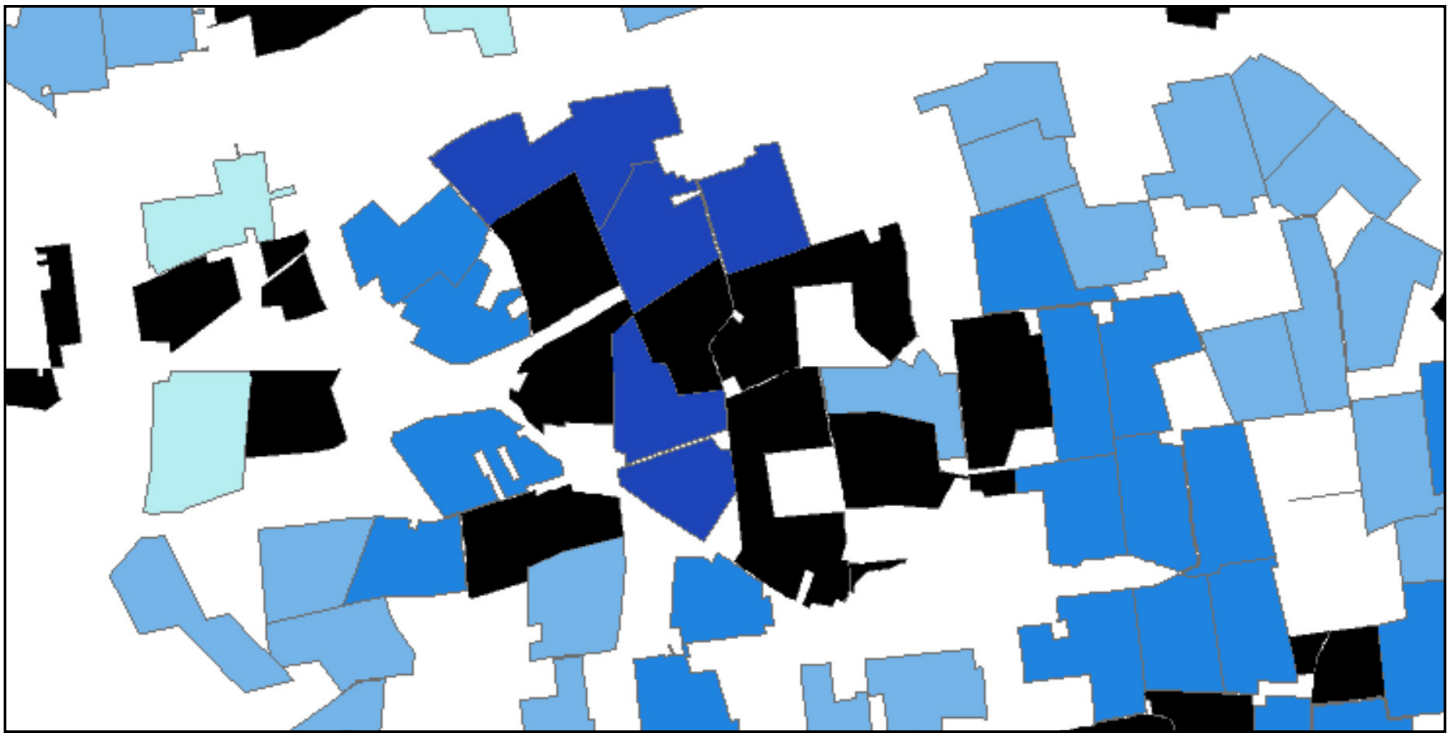
Algorithm Results

The two algorithms produce somewhat different recommendations. Please note that after the scores are calculated, they are mapped to a relative scale of 1-100 (100 belonging to the highest ranking parcel).

Greedy Scoring:



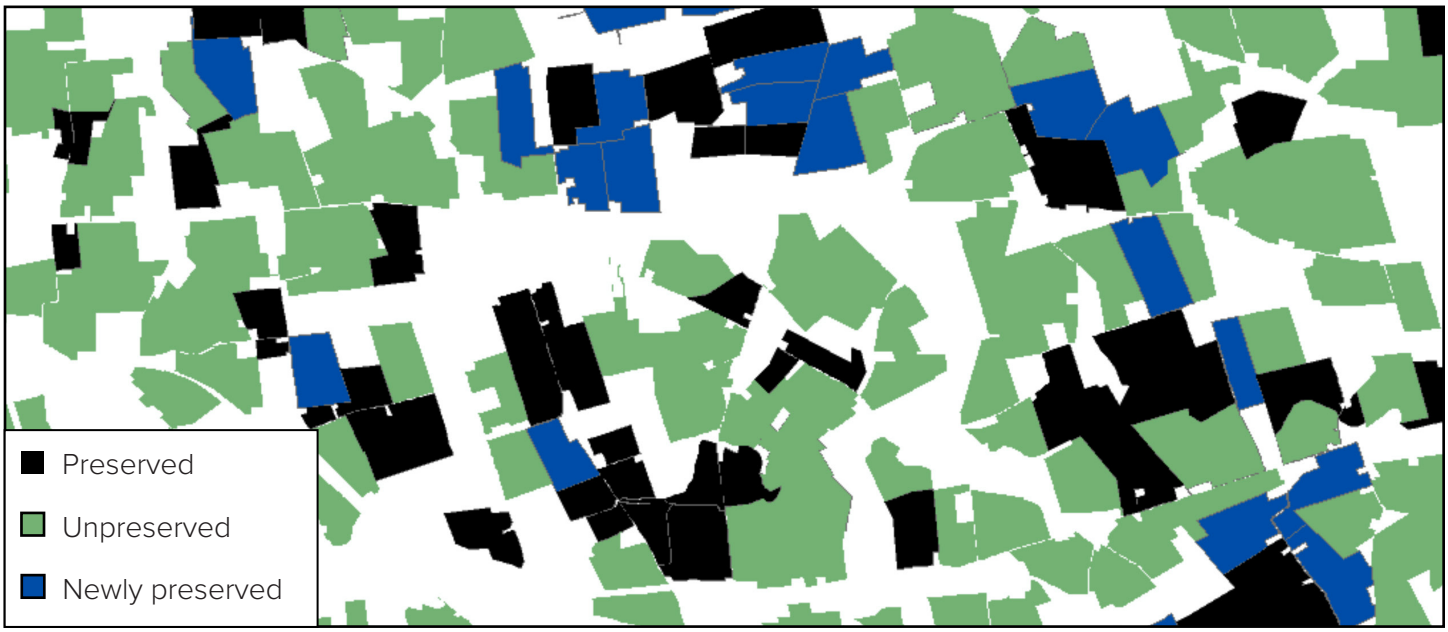
Patient Scoring:



Simulating Outcomes

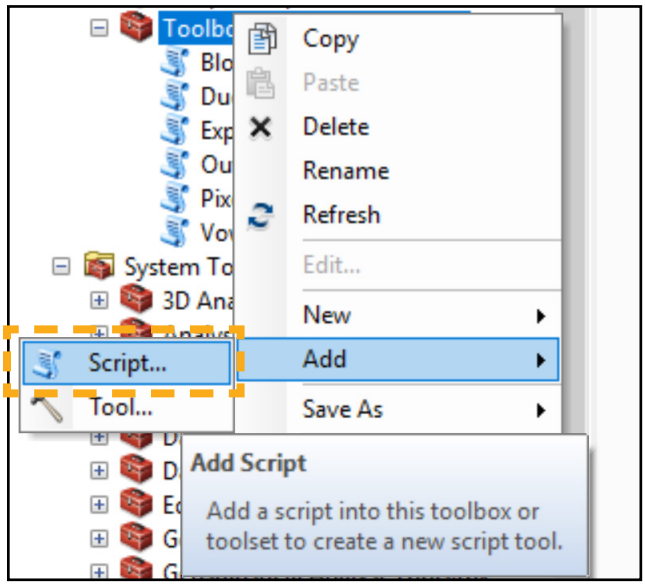
To evaluate the resulting preservation patterns that each algorithm would create in the long-run, the BlobGrower tool includes a "simulating" capability. This capability simulates how the preservation landscape would change over time if the recommendations of the algorithms were followed. This works by running the algorithm for a (user-specified) number of iterations, each time changing the preservation status of a (user-specified) number of the highest ranked parcels. Below are examples of the patterns that emerge. The relative success of the two outcomes could be compared in a separate analysis.

Sample Greedy Simulation Results (15 iterations/15 preservations per iteration):

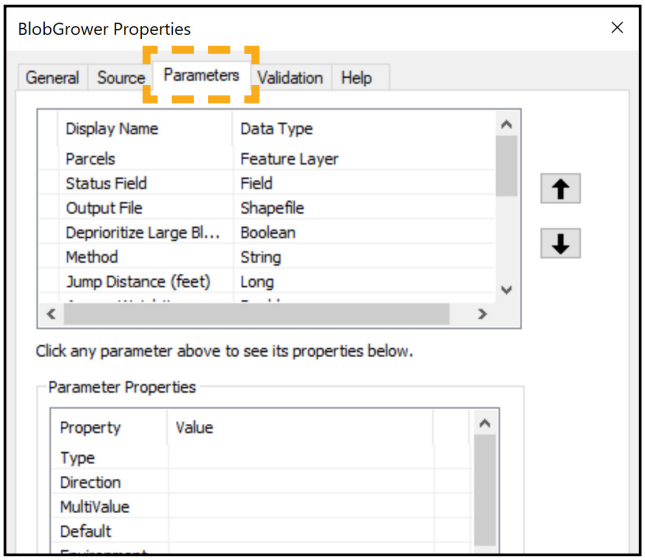


Sample Patient Simulation Results (15 iterations/15 preservations per iteration):

Setting Up BlobGrower in ArcGIS

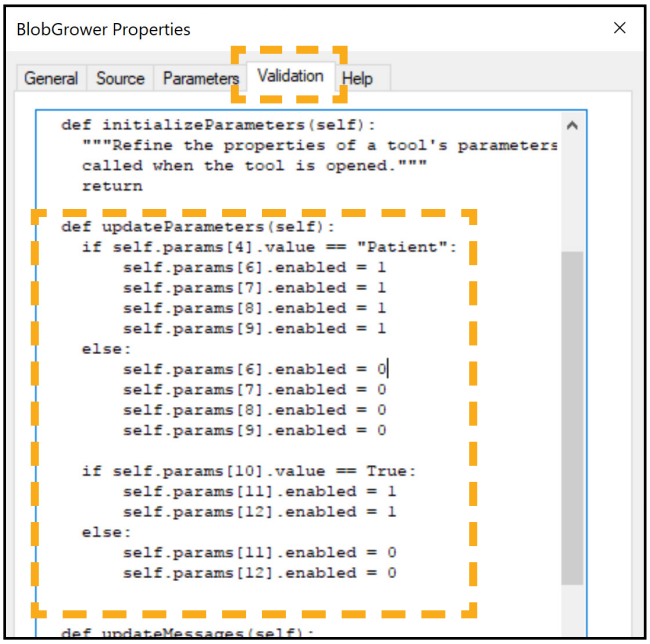


1. Add Script to ArcGIS



2. Configure the following parameters:

DISPLAY NAME	DATA TYPE	PROPERTY>DIRECTION>VALUE
Parcels	Feature Layer	Required>Input
Status_Field	Field	Required>Input
Output_File	Shapefile	Required>Output
Deprioritize_Large_Blobs	Boolean	Optional>Input
Method	String	Required>Input>Filter:ValueList:"Greedy","Patient"
Jump Distance	Double	Required>Input
Access Weight	Double	Required>Input>Default:1
Average Neighbor Size Weight	Double	Required>Input>Default:2
Greedy Score Weight	Double	Required>Input>Default:4
Averaging Iterations	Double	Required>Input>Default:3
Greedy Score Weight	Double	Required>Input>Default:4
Simulate	Boolean	Required>Input
Number of Simulations	Double	Required>Input>Default:1
Parcels to Preserve	Double	Required>Input>Default:5



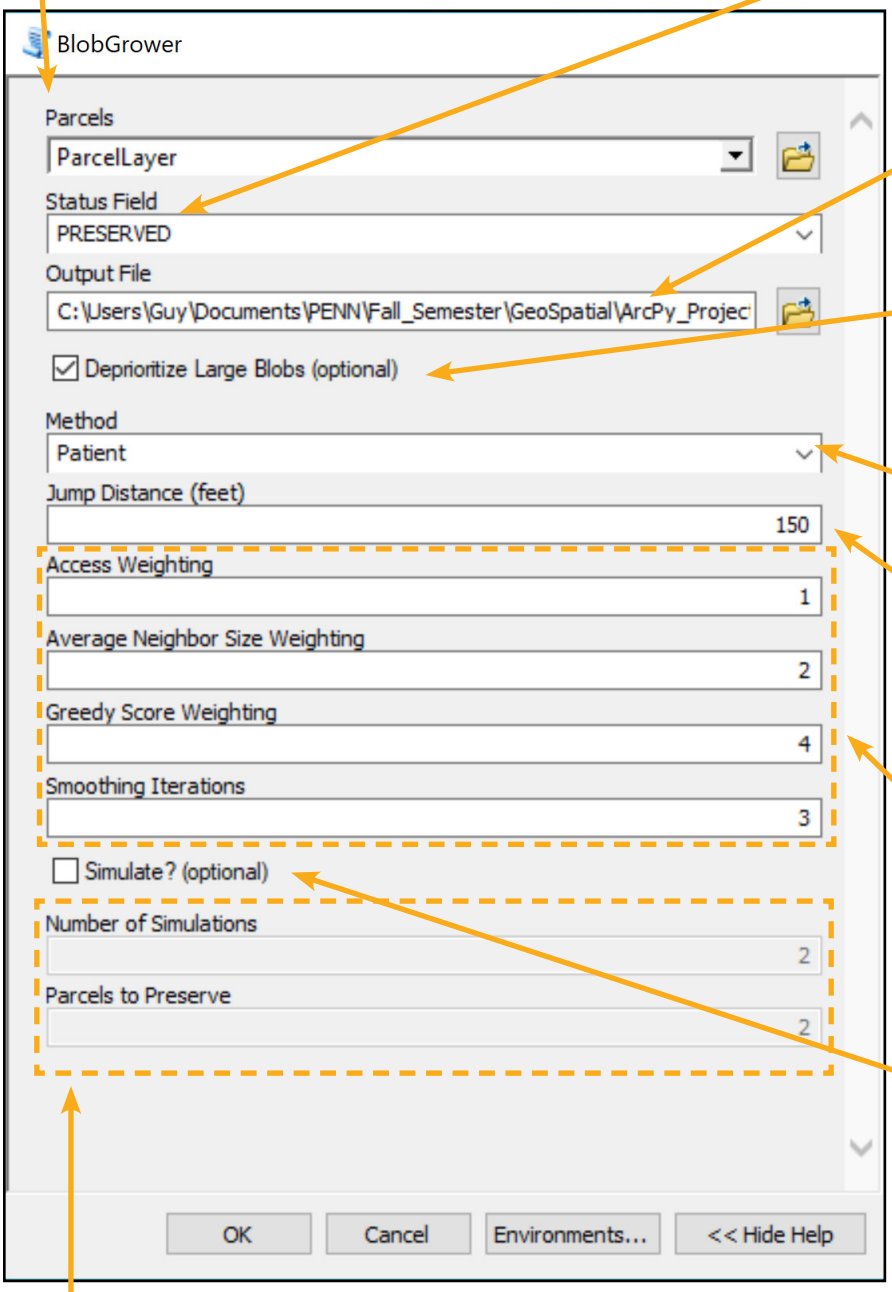
3. Edit the validation script to include the following code:

```
def updateParameters(self):
    if self.params[4].value == "Patient":
        self.params[6].enabled = 1
        self.params[7].enabled = 1
        self.params[8].enabled = 1
        self.params[9].enabled = 1
    else:
        self.params[6].enabled = 0
        self.params[7].enabled = 0
        self.params[8].enabled = 0
        self.params[9].enabled = 0

    if self.params[10].value == True:
        self.params[11].enabled = 1
        self.params[12].enabled = 1
    else:
        self.params[11].enabled = 0
        self.params[12].enabled = 0
```

Using BlobGrower in ArcGIS

To use the BlobGrower tool, import the script into ArcGIS and set up the parameters and validations as instructed in the code header. After doing so, the BlobGrower tool will require the following user inputs in order to execute:



Shapefile of parcels to be analyzed. Must contain both preserved and unpreserved parcels.

Data field which contains each parcel's preservation status.

Output file location

Reduces score of parcels making blobs already larger than 250 acres

Method selector

Longest distance for considering parcels as "neighbors"

Weights for patient method algorithm. Only activated if patient method is selected.

Activate simulation mode. Please note that simulating results (especially when using the patient method) can take a long time to complete processing.

Simulation Settings. Only available if simulation mode is activated.

Appendix: Code (Also Available Here)

BlobGrowerFinal.py

```
"""
SCRIPT: BlobGrower

THIS SCRIPT INCLUDES TWO ALGORITHMS - PATIENT AND GREEDY - FOR THE RANKING OF FARMLAND PARCELS BASED ON THEIR
STRATEGIC ATTRACTIVENESS FOR PRESERVATION. THE SCRIPT ALSO ALLOWS FOR SIMULATING PRESERVATION OUTCOMES IF THE
ALGORITHMS ARE FOLLOWED. PLEASE READ "ARCPY ALGORITHMS FOR STRATEGIC FARMLAND PRESERVATION" REPORT FOR MORE
INFORMATION.

To create an ArcToolbox tool with which to execute this script, do the following.
1   In  ArcMap > Catalog > Toolboxes > My Toolboxes, either select an existing toolbox
    or right-click on My Toolboxes and use New > Toolbox to create (then rename) a new one.
2   Drag (or use ArcToolbox > Add Toolbox to add) this toolbox to ArcToolbox.
3   Right-click on the toolbox in ArcToolbox, and use Add > Script to open a dialog box.
4   In this Add Script dialog box, use Label to name the tool being created, and press Next.
5   In a new dialog box, browse to the .py file to be invoked by this tool, and press Next.
6   In the next dialog box, specify the following inputs (using dropdown menus wherever possible)
    before pressing OK or Finish.
        DISPLAY NAME                    DATA TYPE           PROPERTY>DIRECTION>VALUE
        Parcels                         Feature Layer       Required>Input
        Status_Field                    Field               Required>Input
        Output_File                     Shapefile           Required>Output
        Deprioritize_Large_Blobs        Boolean             Optional>Input
        Method                          String              Required>Input>Filter:ValueList:"Greedy","Patient"
        Jump Distance                   Double              Required>Input
        Access Weight                   Double              Required>Input>Default:1
        Average Neighbor Size Weight    Double              Required>Input>Default:2
        Greedy Score Weight             Double              Required>Input>Default:4
        Averaging Iterations            Double              Required>Input>Default:3
        Greedy Score Weight             Double              Required>Input>Default:4
        Simulate                        Boolean             Required>Input
        Number of Simulations           Double              Required>Input>Default:1
        Parcels to Preserve             Double              Required>Input>Default:5

7   Additionally, include the following validation script in the Validation tab within the Properties Screen:
        def updateParameters(self):
            if self.params[4].value == "Patient":
                self.params[6].enabled = 1
                self.params[7].enabled = 1
                self.params[8].enabled = 1
                self.params[9].enabled = 1
            else:
                self.params[6].enabled = 0
                self.params[7].enabled = 0
                self.params[8].enabled = 0
                self.params[9].enabled = 0

            if self.params[10].value == True:
                self.params[11].enabled = 1
                self.params[12].enabled = 1
            else:
                self.params[11].enabled = 0
                self.params[12].enabled = 0

8       To later revise any of this, right-click to the tool's name and select Properties.
"""


# Import necessary modules
import sys, arcpy, traceback
import arcpy.cartography as CA

# Allow output file to overwrite any existing file of the same name
arcpy.env.overwriteOutput = True

# defines required functions
# translate function maps raw score outputs to 1-100
def translate(value, oldMin, oldMax, newMin, newMax):
    # Figure out how 'wide' each range is
    leftSpan = oldMax - oldMin
    rightSpan = newMax - newMin

    # Convert the left range into a 0-1 range (float)
    valueScaled = float(value - oldMin) / float(leftSpan)

    # Convert the 0-1 range into a value in the right range.
    return newMin + (valueScaled * rightSpan)

# preserve function simulates preserving the parcels with the highest score
def preserve(ScoreField):
    arcpy.AddMessage("Preserving " + str(NumPreserve) + " Parcels")
    # list all scores and take the highest ones (based on user-specified number)
    FieldValues = arcpy.da.SearchCursor(UnpreservedParcels_Output, ScoreField)
    ValuesList = list(FieldValues)
    ValuesList = sorted(ValuesList, reverse=True)
    ToPreserve = ValuesList[NumPreserve][0]

    cur = arcpy.UpdateCursor(UnpreservedParcels_Output)
    Values = arcpy.SearchCursor(UnpreservedParcels_Output)
    CurrentRow = Values.next()

    # For each record, if the score is higher than the cutoff, set status field to 1
    for nextRecord in cur:
        Score = CurrentRow.getValue(ScoreField)
        if Score >= ToPreserve:
            nextRecord.setValue(StatusField, 1)
        cur.updateRow(nextRecord)
        # Move to next row in table
        CurrentRow = Values.next()

    # Delete row and update cursor objects to avoid locking attribute table
    del cur
    del CurrentRow

```

```
try:

    # obtain user input
    ParcelFeatures = arcpy.GetParameterAsText(0)
    arcpy.AddMessage('\n' + "Parcel shapefile: " + ParcelFeatures)

    StatusField = arcpy.GetParameterAsText(1)
    arcpy.AddMessage("The preservation status field is " + StatusField)

    UnpreservedParcels_Output = arcpy.GetParameterAsText(2)
    arcpy.AddMessage("The output shapefile name is " + UnpreservedParcels_Output)

    DePrioritizedChecked = arcpy.GetParameterAsText(3)

    Method = arcpy.GetParameterAsText(4)

    Jump = int(arcpy.GetParameterAsText(5))

    AccessWeight = int(arcpy.GetParameterAsText(6))

    AverageNeighbWeight = int(arcpy.GetParameterAsText(7))

    GreedyScoreWeight = int(arcpy.GetParameterAsText(8))

    AveragingIterations = int(arcpy.GetParameterAsText(9))

    simulate = arcpy.GetParameterAsText(10)

    NumSimulations = max(int(arcpy.GetParameterAsText(11)),1)

    NumPreserve = int(arcpy.GetParameterAsText(12))

    # if not simulating, only iterate over code once
    if str(simulate) != "true":
        NumSimulations = 1

    # begin counting number of times simulation has run
    simcount = 0

    # begin iterations
    for i in range(0,NumSimulations):

        # if currently in the middle of simulation. Set output layer from previous iteration to new input layer.
        if str(simulate) == "true" and simcount > 0:
            arcpy.AddMessage("Simulating iteration " + str(i+1))
            arcpy.DeleteField_management(UnpreservedParcels_Output, ['PatientScr','GreedyScr'])
            NewParcelFeatures = arcpy.CreateFeatureclass_management('in_memory', 'newparcels', 'POLYGON')
            arcpy.CopyFeatures_management(UnpreservedParcels_Output, NewParcelFeatures)
            arcpy.MakeFeatureLayer_management(NewParcelFeatures, 'ParcelLayer')

        # if not simulating or on first simulation iteration, use user-specified input layer
        elif str(simulate) == "true" and simcount == 0:
            arcpy.AddMessage("Simulating iteration " + str(i + 1))
            arcpy.MakeFeatureLayer_management(ParcelFeatures, 'ParcelLayer')

        else:
            arcpy.MakeFeatureLayer_management(ParcelFeatures, 'ParcelLayer')

        # create layers in memory as placeholder for output
        PreservedParcels = arcpy.CreateFeatureclass_management('in_memory', 'prsrvd', 'POLYGON')
        PreservedParcels_Output = arcpy.CreateFeatureclass_management('in_memory', 'prsrvdout', 'POLYGON')

        # separate preserved and unpreserved parcels for separate analyses
        arcpy.SelectLayerByAttribute_management('ParcelLayer', 'NEW_SELECTION', StatusField + " = 1")
        arcpy.CopyFeatures_management('ParcelLayer', PreservedParcels)

        arcpy.SelectLayerByAttribute_management('ParcelLayer', 'NEW_SELECTION', StatusField + " = 0")
        arcpy.CopyFeatures_management('ParcelLayer', UnpreservedParcels_Output)

        # calculate "blob" size of preserved parcels
        CA.AggregatePolygons(PreservedParcels, PreservedParcels_Output, Jump, "", "", "ORTHOGONAL", "", "")
        arcpy.AddGeometryAttributes_management(PreservedParcels_Output, "AREA", "", "ACRES")
        arcpy.AddGeometryAttributes_management(UnpreservedParcels_Output, "AREA", "", "ACRES")

        # set deprioritization factors if user chose to deprioritize large blobs
        if str(DePrioritizedChecked) == "true":
            factor500 = 0
            factor250 = 0.5
        else:
            factor500 = 1
            factor250 = 1

        # create new field to store blob size
        arcpy.AddField_management(PreservedParcels_Output, "WeightedAr", "FLOAT")

        # Create an enumeration of updatable records from the shapefile's attribute table
        cur = arcpy.UpdateCursor(PreservedParcels_Output)

        # Start cursor at the beginning of the attribute table
        Values = arcpy.SearchCursor(PreservedParcels_Output)
        CurrentRow = Values.next()

        # calculate blob size while taking into account the deprioritization factors
        for nextRecord in cur:
            CurrentValue = CurrentRow.getValue("POLY_AREA")
            if CurrentValue >= 500:
                nextRecord.setValue("WeightedAr", CurrentValue * factor500)
                cur.updateRow(nextRecord)
            elif CurrentValue >= 250 and CurrentValue < 500 >= 250:
                nextRecord.setValue("WeightedAr", CurrentValue * factor250)
                cur.updateRow(nextRecord)
            else:
                nextRecord.setValue("WeightedAr", CurrentValue)
                cur.updateRow(nextRecord)
            # Move to next row in table
            CurrentRow = Values.next()

        # Delete row and update cursor objects to avoid locking attribute table
        del cur
        del CurrentRow
```

```
        del CurrentRow

        # create placeholder tables for upcoming calculations
        near_table = arcpy.management.CreateTable('in_memory', 'near_table')
        sum_table = arcpy.management.CreateTable('in_memory', 'sum_table')

        # calculate area sum of nearby preserved blobs for each unpreserved parcel.
        arcpy.GenerateNearTable_analysis(UnpreservedParcels_Output, PreservedParcels_Output, near_table, Jump, "", "", "ALL", "0", "PLANAR")
        arcpy.JoinField_management(near_table, "NEAR_FID", PreservedParcels_Output, "FID", "WeightedAr")
        arcpy.Statistics_analysis(near_table, sum_table, [["WeightedAr", "SUM"]], "IN_FID")
        arcpy.JoinField_management(UnpreservedParcels_Output, "FID", sum_table, "IN_FID", "SUM_WEIGHTEDAR")

        # combine the nearby blob area to the size of the unpreserved parcel and store in new combined acre field
        arcpy.AddField_management(UnpreservedParcels_Output, "CombndAcre", "FLOAT")
        arcpy.CalculateField_management(UnpreservedParcels_Output, "CombndAcre", "!SUM_WEIGHT! + !POLY_AREA!", "PYTHON_9.3","")

        # create "greedy weight" field to store upcoming information
        arcpy.AddField_management(UnpreservedParcels_Output, "GreedyWght", "FLOAT")

        cur = arcpy.UpdateCursor(UnpreservedParcels_Output)
        Values = arcpy.SearchCursor(UnpreservedParcels_Output)
        CurrentRow = Values.next()

        # loop through records and assign a "greedy weight" by adding the parcel's own value and the surrounding
        # "combined value".
        for nextRecord in cur:
            ParcelValue = CurrentRow.getValue("POLY_AREA")
            CombinedValue = CurrentRow.getValue("CombndAcre")
            SurroundingValue = CurrentRow.getValue("SUM_WEIGHT")
            if SurroundingValue == 0:
                nextRecord.setValue("GreedyWght", ParcelValue)
            else:
                nextRecord.setValue("GreedyWght", CombinedValue + ParcelValue)
            cur.updateRow(nextRecord)
            # Move to next row in table
            CurrentRow = Values.next()

        # only perform if method is set to greedy
        if Method == "Greedy":

            # create final "greedy score" field
            arcpy.AddField_management(UnpreservedParcels_Output, "GreedyScr", "INTEGER")

            # Delete row and update cursor objects to avoid locking attribute table
            del cur
            del CurrentRow

            FieldValues = arcpy.da.SearchCursor(UnpreservedParcels_Output, "GreedyWght")
            ValuesList = list(FieldValues)
            MaxSize = max(ValuesList)
            MinSize = min(ValuesList)

            cur = arcpy.UpdateCursor(UnpreservedParcels_Output)
            Values = arcpy.SearchCursor(UnpreservedParcels_Output)
            CurrentRow = Values.next()

            # loop through records and translate greedy weight to greedy score from 1-100
            for nextRecord in cur:
                WeightValue = CurrentRow.getValue("GreedyWght")
                NewValue = translate(WeightValue, MinSize[0], MaxSize[0], 1, 100)
                nextRecord.setValue("GreedyScr", NewValue)
                cur.updateRow(nextRecord)
                # Move to next row in table
                CurrentRow = Values.next()

            # Delete row and update cursor objects to avoid locking attribute table
            del cur
            del CurrentRow

            # if user chose to simulate, change preservation status of highest ranked parcels
            if str(simulate) == "true":
                preserve("GreedyScr")



        else:
            arcpy.AddMessage("This might take a while...")

            # create empty tables in memory for future calculations
            near_table2 = arcpy.management.CreateTable('in_memory', 'near_table2')
            sum_table2 = arcpy.management.CreateTable('in_memory', 'sum_table2')

            # calculate the total area to which each parcel has access (total area of neighboring parcels)
            arcpy.GenerateNearTable_analysis(UnpreservedParcels_Output, UnpreservedParcels_Output, near_table2, Jump, "", "", "ALL", "0", "PLANAR")
            arcpy.JoinField_management(near_table2, "NEAR_FID", UnpreservedParcels_Output, "FID", "POLY_AREA")
            arcpy.Statistics_analysis(near_table2, sum_table2, [["POLY_AREA", "SUM"],["NEAR_FID", "COUNT"]], "IN_FID")
            arcpy.JoinField_management(UnpreservedParcels_Output, "FID", sum_table2, "IN_FID", "SUM_POLY_AREA")
            arcpy.AddField_management(UnpreservedParcels_Output, "NeighbArea", "FLOAT")
            arcpy.CalculateField_management(UnpreservedParcels_Output, "NeighbArea", "!SUM_POLY_A!", "PYTHON_9.3", "")

            # calculate the average size of each parcel's neighboring parcels
            arcpy.JoinField_management(UnpreservedParcels_Output, "FID", sum_table2, "IN_FID", "COUNT_NEAR_FID")

            arcpy.AddField_management(UnpreservedParcels_Output, "AvgNeighSz", "FLOAT")
            arcpy.AddField_management(UnpreservedParcels_Output, "PatientWgt", "FLOAT")

            cur = arcpy.UpdateCursor(UnpreservedParcels_Output)
            Values = arcpy.SearchCursor(UnpreservedParcels_Output)
            CurrentRow = Values.next()

            for nextRecord in cur:
                SizeNeighbors = CurrentRow.getValue("NeighbArea")
                NumNeighbors = CurrentRow.getValue("COUNT_NEAR")
                if NumNeighbors == 0:
                    nextRecord.setValue("AvgNeighSz", 0)
                else:
                    nextRecord.setValue("AvgNeighSz", SizeNeighbors / NumNeighbors)

                Combined = CurrentRow.getValue("GreedyWght")
                AverageNeighbs = CurrentRow.getValue("AvgNeighSz")
```

```
                # set the patient weight value by adding all calculated score and weighing them by user specified amounts
                nextRecord.setValue("PatientWgt", (SizeNeighbors * AccessWeight) + (AverageNeighbs * AverageNeighbWeight) + (Combined * GreedyScoreWeight))
                cur.updateRow(nextRecord)
                CurrentRow = Values.next()

            # Delete row and update cursor objects to avoid locking attribute table
            del cur
            del CurrentRow

            # create empty tables in memory for future calculations
            near_table3 = arcpy.management.CreateTable('in_memory', 'near_table3')
            sum_table3 = arcpy.management.CreateTable('in_memory', 'sum_table3')

            arcpy.AddField_management(UnpreservedParcels_Output, "LocalValue", "FLOAT")

            # based on user-specified number of iterations, give each parcel the average patient weight of the surrounding parcels
            for i in range(AveragingIterations):
                arcpy.AddMessage("Averaging iteration " + str(i + 1))
                arcpy.GenerateNearTable_analysis(UnpreservedParcels_Output, UnpreservedParcels_Output, near_table3, Jump, "","", "ALL", "0", "PLANAR")
                arcpy.JoinField_management(near_table3, "NEAR_FID", UnpreservedParcels_Output, "FID", "PatientWgt")
                arcpy.Statistics_analysis(near_table3, sum_table3, [["PatientWgt", "MEAN"]], "IN_FID")
                arcpy.JoinField_management(UnpreservedParcels_Output, "FID", sum_table3, "IN_FID", "MEAN_PatientWgt")
                arcpy.CalculateField_management(UnpreservedParcels_Output, "PatientWgt", "(!MEAN_Patie!+!PatientWgt!)/2", "PYTHON_9.3", "")
                arcpy.DeleteField_management(UnpreservedParcels_Output, "MEAN_Patie")

                arcpy.CalculateField_management(UnpreservedParcels_Output, "LocalValue", "!LocalValue! + !PatientWgt!", "PYTHON_9.3", "")

            arcpy.AddField_management(UnpreservedParcels_Output, "PatientScr", "INTEGER")

            FieldValues = arcpy.da.SearchCursor(UnpreservedParcels_Output, "LocalValue")
            ValuesList = list(FieldValues)
            MaxSize = max(ValuesList)
            MinSize = min(ValuesList)

            cur = arcpy.UpdateCursor(UnpreservedParcels_Output)
            Values = arcpy.SearchCursor(UnpreservedParcels_Output)
            CurrentRow = Values.next()

            # Map each patient weight to a patient score from 1-100
            for nextRecord in cur:
                WeightValue = CurrentRow.getValue("LocalValue")
                NewValue = translate(WeightValue, MinSize[0], MaxSize[0], 1, 100)
                nextRecord.setValue("PatientScr", NewValue)
                cur.updateRow(nextRecord)
                # Move to next row in table
                CurrentRow = Values.next()

            # Delete row and update cursor objects to avoid locking attribute table
            del cur
            del CurrentRow

            # if user chose to simulate, change preservation status of highest ranked parcels
            if str(simulate) == "true":
                preserve("PatientScr")

        # delete all interim calculated fields
        arcpy.DeleteField_management(UnpreservedParcels_Output, ['LocalValue','PatientWgt','AvgNeighSz','COUNT_NEAR'
            ,'NeighbArea','SUM_POLY_A','GreedyWght','CombndAcre','SUM_Weight','POLY_AREA'])

        # create a placeholder shapefile in memory and re-merge the preserved and unpreserved parcels
        UnpreservedPlaceholder = arcpy.CreateFeatureclass_management('in_memory', 'unprsrvdPH', 'POLYGON')
        arcpy.CopyFeatures_management(UnpreservedParcels_Output, UnpreservedPlaceholder)
        arcpy.Merge_management([UnpreservedPlaceholder, PreservedParcels], UnpreservedParcels_Output)

        # count the number of simulations
        simcount = simcount + 1

        # delete all items created in memory
        arcpy.Delete_management("in_memory")


    # Add a layer for that new shapefile to the active data frame
    currentMap = arcpy.mapping.MapDocument("CURRENT")
    currentDataFrame = currentMap.activeDataFrame
    layerToBeDisplayed = arcpy.mapping.Layer(UnpreservedParcels_Output)
    arcpy.mapping.AddLayer(currentDataFrame, layerToBeDisplayed, "TOP")
    del currentMap


except Exception as e:
    # If unsuccessful, end gracefully by indicating why
    arcpy.AddError('\n' + "Script failed because: \t\t" + e.message)
    # ... and where
    exceptionreport = sys.exc_info()[2]
    fullermessage = traceback.format_tb(exceptionreport)[0]
    arcpy.AddError("at this location: \n\n" + fullermessage + "\n")
```