Code for primary analysis was generalized as follows:

```
#-------------------------------------------------------------------------------
# Name:       Least Cost Analysis
# Purpose:     Take a list of locations from a shapefile and calculate to and
#              fro least cost paths that could be turned into time estimates
#              for travel between these places.
# Author:      ciszka
#
# Created:     04/02/2020, 29 Sep 2020, 6 May 2021
# Copyright:   (c) ciszka 2020
# Licence:     MIT see MIT license file
#-------------------------------------------------------------------------------

import os
import arcpy, sys #import python modules for arcGIS
import math
from arcpy import env
from arcpy.sa import *

arcpy.CheckOutExtension("Spatial")
arcpy.CheckOutExtension("3D")
arcpy.CheckOutExtension("Math")
arcpy.env.overwriteOutput = True

#Set the workspace for all these things to go into
env.workspace = r"" #Put your file path here between the "" with forward slashes replacing backslashes
saveWorkSpace = r"" #Put your file path here between the "" with forward slashes replacing
backslashes, this should be the same as above unless you want to save to a different locations for one
reason

#Drives where already created data lives, some of these points are optional depending on how data is
structured locally
#drive = "D" #this is just the name of the drive
DEM = env.workspace + "/" + "" #Between the last "" put the name of the DEM you'd like to use, this can
be used to calculate surfaces for slope and cost
resolution = 10.0 #Set this to the actual resolution of your DEM data a float (a number with a decimal
point) value

greatHouse = saveWorkSpace + "/" + "" #This will be the list of sites you want to use for the analysis

#Slope can either be set to an established slope or created
#slope = "C:/Users/Wills/Documents/ArcGIS/LCA.gdb/slope_Temp" #this is a set slope
#create the surfaces we want to use if using the whole walk through below calculates the slope within
this program
newSlopeName = ""#This is the new name for the slope we may calculate for use in the future, this
needs to be a whole path including the geodatabase this wants to be part of
slope = arcpy.gp.Slope_sa(DEM, newSlopeName, "DEGREE", "1", "PLANAR", "METER")
```

```
#Cost can either be set to an established cost surface or calculated
#Cost calculation for this paper arcpy.gp.RasterCalculator_sa('(10.0/1000.0) / (6.0 * Exp(-
3.5*Abs(Tan(("[INSERT SLOPE RASTER HERE]"*3.14159)/180.0)+0.05)))', "") #Between the last "" you
need to put a whole pathway with a name separated by foward slashes
#cost = "" #this is a set cost given as a file path separated by forward slashes
#Cost calculation, must be done as a raster math outside of arcpy and the following uses time in hours
from White 2015
newCostName = saveWorkSpace + "/tempCost" #This is the name for the cost raster that we may
calculate for use in the future
numerator = float(resolution/1000.0)
inRaster01 = "slope_Temp" # or as set above as required
newSlope = Times(inRaster01,float(math.pi))
divSlope = Divide(newSlope,180.0)
tanSlope = Tan(divSlope)
plusSlope = Plus(tanSlope,0.05)
absSlope = abs(plusSlope)
timesSlope = Times(-3.5,absSlope)
expoSlope = Exp(timesSlope)
denominator = Times(6.0,expoSlope)
cost = Divide(numerator,denominator) #Cost in Hours

#Names for the newly generated tables and paths file with the names that can be changed
timesTable = saveWorkSpace + "/timesTable.dbf"
pathsFile = saveWorkSpace + "/regionalCostPaths"

counter = 0

houseMeasureFrom = []
houseMeasureTo = []
houseNames = []
houseSize = []

def cleanSlate(fileName): #removes all the temporary files
if arcpy.Exists(fileName): #this first one is for the layer that has just the individual photo point we are
looking at
        arcpy.Delete_management(fileName)
else:
pass

def addAndCalcTextField(tableToAdd,newField,fillField): #I add fields to a new thing multiple times and
want to fill those with particular values
   arcpy.AddField_management(tableToAdd, newField, "TEXT", field_length=50) #The below adds
identification information to the particular line so that it has ID information for where it measures from
and where it will measure to
   arcpy.CalculateField_management(tableToAdd, newField, fillField, "PYTHON")       #I want to add the
photo name to the row that the the particular point is from, could
```

```
#Need to create search cursor to get the OBJECT ID for all the GH in the list
rows = arcpy.SearchCursor(greatHouse)
for row in rows:
   houseMeasureFrom.append(row.OBJECTID)
   houseMeasureTo.append(row.OBJECTID)
   houseNames.append(row.LANumber)
   houseSize.append(row.RankGroup)

try:
   #Make the cost surface with a particular thing
   myTempLayers = ["timesTable", "regionalCostPaths"] #Removes previous versions of the files that
were ran while testing etc.
   for layer in range(len(myTempLayers)):
      cleanSlate(myTempLayers[layer])
   myTempLayers = ["tempCostPath", "MyTempLine", "tempReclass", "tempTable"]
   for fro in range(len(houseMeasureFrom)): #For loop 2 will calculate the cost paths from each raster to
the other locations
      print ("I'm in the first loop" + str(houseMeasureFrom[fro])) # This version requires the computer to
have enough spacec to hhold all the ACS and BKLNK files FYI
      curACS = saveWorkSpace + "/ACS_" + str(houseMeasureFrom[fro])
      curBkLnk = saveWorkSpace + "/bklnk_" + str(houseMeasureFrom[fro]) # Must use real numbers
      segWhere = '"OBJECTID" = %s' % houseMeasureFrom[fro] #Creates the SQL select to make the
feature for one location
      layerNameFrom = str(houseMeasureFrom[fro]) #Turns the ObjectID into a string so it can be used as
such in future naming strings
      tempLayer = arcpy.MakeFeatureLayer_management(greatHouse, layerNameFrom, segWhere)
#creates a temporary feature class with just that location
      arcpy.gp.CostDistance_sa(tempLayer, cost, curACS, "", curBkLnk, "", "", "", "", "") #This actually runs
the accumulated cost surface calculation
      to = fro
      for to in range(fro, len(houseMeasureTo)-1):
         to = to + 1
         destination = str(houseMeasureFrom[to]) #sets the destination to a string value for the purposes
of naming
         print ("I'm in the second loop"+ destination)
         counter = counter + 1 #Counter to check if we're making the first path which influences how we
save the data
         segWhereTo = '"OBJECTID" = %s' % houseMeasureTo[to] #Creates the SQL select to make the
feature for one location
         tempLayerTo = arcpy.MakeFeatureLayer_management(greatHouse, destination, segWhereTo)
#creates a temporary feature class with just that location
         toFroRaster = saveWorkSpace + "/tempCostPath" #This it the name for the cost path that we will
use
         arcpy.gp.CostPath_sa(tempLayerTo, curACS, curBkLnk, toFroRaster, "EACH_CELL", "UTM_ZONE")
# This creates the temporary cost path based on where we are going
         costOut = arcpy.GetRasterProperties_management(toFroRaster, "MAXIMUM") # THis lets us
identify the highest value within that cost path
         reclassPath = saveWorkSpace + "/tempReclass"
```

```
        reclassValues = "1 " + costOut.getOutput(0) + " 1" #This sets the values for the reclassification to all being one; it gets the maximum value within the cost path
        arcpy.gp.Reclassify_sa(toFroRaster, "Value", reclassValues, reclassPath, "NODATA") #this will produce one line for each path by reclassing all segments
        arcpy.RasterToPolyline_conversion(in_raster=reclassPath, out_polyline_features=saveWorkSpace + "/MyTempLine", background_value="NODATA", minimum_dangle_length="0", simplify="SIMPLIFY", raster_field="VALUE")                #Raster to polyline
        addAndCalcTextField("MyTempLine", "GHFrom","'" + str(houseNames[fro]) + "'")
        addAndCalcTextField("MyTempLine", "GHFromSize","'" + str(houseSize[fro]) + "'")
        addAndCalcTextField("MyTempLine", "GHTo","'" + str(houseNames[to]) + "'")
        addAndCalcTextField("MyTempLine", "GHToSize","'" + str(houseSize[to]) + "'")
        addAndCalcTextField("MyTempLine", "FromToSize","'" + str(houseSize[fro]) + str(houseSize[to]) + "'")
        arcpy.AddField_management("MyTempLine", "PathLengthMiles", "FLOAT") #The below adds identification information to the particular line so that it has ID information for where it measures from and where it will measure to
        outTable = saveWorkSpace + "/tempTable"
        arcpy.gp.ZonalStatisticsAsTable_sa(reclassPath, "Value", cost, outTable, "DATA", "ALL")
        addAndCalcTextField(outTable, "GHFrom","'" + str(houseNames[fro]) + "'")
        addAndCalcTextField(outTable, "GHTo","'" + str(houseNames[to]) + "'")
        if counter == 1:
            pathsFile = arcpy.Rename_management("MyTempLine", "regionalCostPaths", "FeatureClass") # makes the original feature class to append the other paths to
            timesTable = arcpy.Rename_management(in_data=outTable, out_data=saveWorkSpace + "/timesTable", data_type="Table") # makes the original table to attach the times information to
        else:
            arcpy.Append_management("MyTempLine", pathsFile, "NO_TEST") #This feature will need to be saved in a growing shapefile for the paths
            arcpy.Append_management(outTable, timesTable, "NO_TEST") #This feature will need to be saved in a growing table for the times
        for layer in range(len(myTempLayers)):
            cleanSlate(myTempLayers[layer])
    #Do a final join between the two final things and export that
    joinLayer = "joinLayerTemp"
    finalLayerName = saveWorkSpace + "/pathTimeCombined"
    segWhere =  '"OBJECTID" IS NOT NULL'
    arcpy.MakeFeatureLayer_management(pathsFile, joinLayer, segWhere)
    arcpy.AddJoin_management(joinLayer, "OBJECTID", timesTable, "OBJECTID")
    arcpy.CopyFeatures_management(joinLayer, finalLayerName)
    arcpy.AddField_management(finalLayerName, "PathLengthMiles", "FLOAT")
    arcpy.AddGeometryAttributes_management(Input_Features=finalLayerName, Geometry_Properties="LENGTH", Length_Unit="MILES_US", Area_Unit="", Coordinate_System="")

except arcpy.ExecuteError: #Tell me what error occurred
    msgs = arcpy.GetMessages(2)
    print (msgs)
    print ("These are the exceptions I threw, sorry")
```

```
finally:
   arcpy.CheckInExtension("Spatial")
   arcpy.CheckInExtension("3D")
   print("I'm done, thank you for your time, have a pleasant pandemic")
```

The code for the secondary analysis is generalized as follows:

```
#-------------------------------------------------------------------------------
# Name:       Least Cost Analysis
# Purpose:     Take a list of locations from a shapefile and calculate to and
#              fro least cost paths that could be turned into time estimates
#              for travel between these places.
# Author:     ciszka
#
# Created:     04/02/2020, 29 Sep 2020, 6 May 2021, 6 Oct 2021
# Copyright:   (c) ciszka 2020
# Licence:     MIT see MIT license file
#-------------------------------------------------------------------------------

import os
import arcpy, sys #import python modules for arcGIS
import math
from arcpy import env
from arcpy.sa import *
from datetime import datetime, date

arcpy.CheckOutExtension("Spatial")
arcpy.CheckOutExtension("3D")
arcpy.CheckOutExtension("Math")
arcpy.env.overwriteOutput = True
date  = datetime.now()
print str(date)

#Set the workspace for all these things to go into
env.workspace = r"" #Put your file path here between the "" with forward slashes replacing backslashes
saveWorkSpace = r"" #Put your file path here between the "" with forward slashes replacing
backslashes, this should be the same as above unless you want to save to a different locations for one
reason

#Drives where already created data lives, some of these points are optional depending on how data is
structured locally
#drive = "D" #this is just the name of the drive
DEM = env.workspace + "/" + "" #Between the last "" put the name of the DEM you'd like to use, this can
be used to calculate surfaces for slope and cost
resolution = 10.0 #set as a float value

greatHouse = saveWorkSpace + "/" + "" #This will be the list of sites you want to use for the analysis
```

```
#Slope can either be set to an established slope or created
newSlopeName = ""#This is the new name for the slope we may calculate for use in the future, this needs to be a whole path including the geodatabase this wants to be part of
#slope = saveWorkSpace + "Slope_CHCU" #this is a set slope

#Cost can either be set to an established cost surface or calculated
#Cost calculation for this paper arcpy.gp.RasterCalculator_sa('(10.0/1000.0) / (6.0 * Exp(-3.5*Abs(Tan(("[INSERT SLOPE RASTER HERE]"*3.14159)/180.0)+0.05)))', "") #Between the last "" you need to put a whole pathway with a name separated by foward slashes
#cost = "" #this is a set cost given as a file path separated by forward slashes
newCostName = saveWorkSpace + "/tempCost" #This is the name for the cost raster that we may calculate for use in the future

#Names for the newly generated tables and paths file with the names that can be changed
timesTable = saveWorkSpace + "/timesTable.dbf"
pathsFile = saveWorkSpace + "/regionalCostPaths"

counter = 0

houseMeasureFrom = []
houseMeasureTo = []
houseNames = []
houseSize = []

def cleanSlate(fileName): #removes all the temporary files
   if arcpy.Exists(fileName): #this first one is for the layer that has just the individual photo point we are looking at
      arcpy.Delete_management(fileName)
   else:
      pass

def addAndCalcTextField(tableToAdd,newField,fillField): #I add fields to a new thing multiple times and want to fill those with particular values
   arcpy.AddField_management(tableToAdd, newField, "TEXT", field_length=50) #The below adds identification information to the particular line so that it has ID information for where it measures from and where it will measure to
   arcpy.CalculateField_management(tableToAdd, newField, fillField, "PYTHON")        #I want to add the photo name to the row that the the particular point is from, could

#Need to create search cursor to get the OBJECT ID for all the GH in the list
rows = arcpy.SearchCursor(greatHouse)
for row in rows:
   houseMeasureFrom.append(row.OBJECTID)
   houseMeasureTo.append(row.OBJECTID)
   houseNames.append(row.LANumber)
   houseSize.append(row.RankGroup)
```

```
#create the surfaces we want to use if using the whole walk through
#Slope calculation
slope = arcpy.gp.Slope_sa(DEM, newSlopeName, "DEGREE", "1", "PLANAR", "METER")

#Cost calculation, must be done as a raster math outside of arcpy
#This version uses the time in hours from White 2015 which looks like this in
#ArcGIS raster calculator:
#'(10.0/1000.0) / (6.0 * Exp(-3.5*Abs(Tan(([INSERT SLOPE RASTER
HERE]*3.14159)/180.0)+0.05)))'costCalc.save("C:/Users/Wills/Documents/ArcGIS/LCA.gdb/tempCost")
numerator = float(resolution/1000.0)
inRaster01 = "slope_Temp" # or as set above as required
newSlope = Times(inRaster01,float(math.pi))
divSlope = Divide(newSlope,180.0)
tanSlope = Tan(divSlope)
plusSlope = Plus(tanSlope,0.05)
absSlope = abs(plusSlope)
timesSlope = Times(-3.5,absSlope)
expoSlope = Exp(timesSlope)
denominator = Times(6.0,expoSlope)
cost = Divide(numerator,denominator) #Cost in Hours

try:
   myTempLayers = ["timesTable", "regionalCostPaths"] #Removes previous versions of the files that
were ran while testing etc.
   for layer in range(len(myTempLayers)):
      cleanSlate(myTempLayers[layer])
   myTempLayers = ["tempCostPath", "MyTempLine", "tempReclass", "tempTable"]
   for fro in range(len(houseMeasureFrom)): #For loop 1 will calculate the accumulated cost surfaces
      print ("I'm in the first loop" + str(houseMeasureFrom[fro])) # This version requires the computer to
have enough space to hold all the ACS and BKLNK files FYI
      curACS = saveWorkSpace + "/ACS_" + str(houseMeasureFrom[fro])
      curBkLnk = saveWorkSpace + "/bklnk_" + str(houseMeasureFrom[fro]) # Must use real numbers else
you get this error (includes above comment"C:\Program Files
(x86)\PyScripter\Lib\rpyc.zip\rpyc\core\stream.py", line 223, in read EOFError: [Errno 10054] An existing
connection was forcibly closed by the remote host
      segWhere = '"OBJECTID" = %s' % houseMeasureFrom[fro] #Creates the SQL select to make the
feature for one location
      layerNameFrom = str(houseMeasureFrom[fro]) #Turns the ObjectID into a string so it can be used as
such in future naming strings
      tempLayer = arcpy.MakeFeatureLayer_management(greatHouse, layerNameFrom, segWhere)
#creates a temporary feature class with just that location
      arcpy.gp.CostDistance_sa(tempLayer, cost, curACS, "", curBkLnk, "", "", "", "", "") #This actually runs
the accumulated cost surface calculation
   for fro in range(len(houseMeasureFrom)): #For loop 2 will calculate the cost paths from each raster to
the other locations
      print ("I'm in the first loop" + str(houseMeasureFrom[fro])) # This version requires the computer to
have enough spacec to hhold all the ACS and BKLNK files FYI
      curACS = saveWorkSpace + "/ACS_" + str(houseMeasureFrom[fro])
```

```
    curBkLnk = saveWorkSpace + "/bklnk_" + str(houseMeasureFrom[fro]) # Must use real numbers else you get this error (includes above comment"C:\Program Files (x86)\PyScripter\Lib\rpyc.zip\rpyc\core\stream.py", line 223, in read EOFError: [Errno 10054] An existing connection was forcibly closed by the remote host
    to = fro
    for to in range(fro, len(houseMeasureTo)-1):
      to = to + 1
      destination = str(houseMeasureFrom[to]) #sets the destination to a string value for the purposes of naming
      print ("I'm in the second loop"+ destination)
      counter = counter + 1 #Counter to check if we're making the first path which influences how we save the data
      segWhereTo = '"OBJECTID" = %s' % houseMeasureTo[to] #Creates the SQL select to make the feature for one location
      tempLayerTo = arcpy.MakeFeatureLayer_management(greatHouse, destination, segWhereTo) #creates a temporary feature class with just that location
      toFroRaster = saveWorkSpace + "/tempCostPath_" + str(houseMeasureFrom[fro]) + str(houseMeasureTo[to])#This it the name for the cost path that we will use
      arcpy.gp.CostPath_sa(tempLayerTo, curACS, curBkLnk, toFroRaster, "EACH_CELL", "UTM_ZONE") # This creates the temporary cost path based on where we are going
      costOut = arcpy.GetRasterProperties_management(toFroRaster, "MAXIMUM") # THis lets us identify the highest value within that cost path
      reclassPath = saveWorkSpace + "/tempReclass"
      reclassValues = "1 " + costOut.getOutput(0) + " 1" #This sets the values for the reclassification to all being one; it gets the maximum value within the cost path
      arcpy.gp.Reclassify_sa(toFroRaster, "Value", reclassValues, reclassPath, "NODATA") #this will produce one line for each path by reclassing all segments
      arcpy.RasterToPolyline_conversion(in_raster=reclassPath, out_polyline_features=saveWorkSpace + "/MyTempLine", background_value="NODATA", minimum_dangle_length="0", simplify="SIMPLIFY", raster_field="VALUE")                #Raster to polyline
      addAndCalcTextField("MyTempLine", "GHFrom","'" + str(houseNames[fro]) + "'")
      addAndCalcTextField("MyTempLine", "GHFromSize","'" + str(houseSize[fro]) + "'")
      addAndCalcTextField("MyTempLine", "GHTo","'" + str(houseNames[to]) + "'")
      addAndCalcTextField("MyTempLine", "GHToSize","'" + str(houseSize[to]) + "'")
      addAndCalcTextField("MyTempLine", "FromToSize","'" + str(houseSize[fro]) + str(houseSize[to]) + "'")
      arcpy.AddField_management("MyTempLine", "PathLengthMiles", "FLOAT") #The below adds identification information to the particular line so that it has ID information for where it measures from and where it will measure to
      outTable = saveWorkSpace + "/tempTable"
      arcpy.gp.ZonalStatisticsAsTable_sa(reclassPath, "Value", cost, outTable, "DATA", "ALL")
      addAndCalcTextField(outTable, "GHFrom","'" + str(houseNames[fro]) + "'")
      addAndCalcTextField(outTable, "GHTo","'" + str(houseNames[to]) + "'")
      if counter == 1:
        pathsFile = arcpy.Rename_management("MyTempLine", "regionalCostPaths", "FeatureClass") # makes the original feature class to append the other paths to
        timesTable = arcpy.Rename_management(in_data=outTable, out_data=saveWorkSpace + "/timesTable", data_type="Table") # makes the original table to attach the times information to
```

```
        else:
            arcpy.Append_management("MyTempLine", pathsFile, "NO_TEST") #This feature will need to
be saved in a growing shapefile for the paths
            arcpy.Append_management(outTable, timesTable, "NO_TEST") #This feature will need to be
saved in a growing table for the times
        for layer in range(len(myTempLayers)):
            cleanSlate(myTempLayers[layer])
    #Do a final join between the two final things and export that
    joinLayer = "joinLayerTemp"
    finalLayerName = saveWorkSpace + "/pathTimeCombined"
    segWhere =  '"OBJECTID" IS NOT NULL'
    arcpy.MakeFeatureLayer_management(pathsFile, joinLayer, segWhere)
    arcpy.AddJoin_management(joinLayer, "OBJECTID", timesTable, "OBJECTID")
    arcpy.CopyFeatures_management(joinLayer, finalLayerName)
    arcpy.AddField_management(finalLayerName, "PathLengthMiles", "FLOAT")
    arcpy.AddGeometryAttributes_management(Input_Features=finalLayerName,
Geometry_Properties="LENGTH", Length_Unit="MILES_US", Area_Unit="", Coordinate_System="")

except arcpy.ExecuteError: #Tell me what error occurred
    msgs = arcpy.GetMessages(2)
    print (msgs)
    print ("These are the exceptions I threw, sorry")

finally:
    date  = datetime.now()
    print str(date)
    arcpy.CheckInExtension("Spatial")
    arcpy.CheckInExtension("3D")
    print("I'm done, thank you for your time, have a pleasant pandemic")
```