Evening Hade

Lab 3, Part 1

For GIS 5571

The majority of this code is the same as that which I used for Part 2 of Lab 2. There will be new code at the end.

Well, I guess it's all new because I never turned Lab 2, Part 2 in.

The first step of this part of the lab will be to get all my imports, set up my ArcGIS project, get my URLs, and assign an output path:

```
#Imports
import os
import requests
import arcpy
import subprocess
from zipfile import ZipFile

#Initial setup
project = arcpy.mp.ArcGISProject("CURRENT")
map = project.listMaps()[0]
ref = arcpy.SpatialReference(26915)  #NAD zone 15N

surfaceUrl =
'https://resources.gisdata.mn.gov/pub/data/elevation/lidar/county/
wabasha/laz/' #Elevation data download link
landcoverUrl =
'https://resources.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_dnr/
biota_landcover_nlcd_mn_2016/tif_biota_landcover_nlcd_mn_2016.zip'
#Land cover data download link

outputPath = r"C:\ArcGIS\Projects\Lab3-1" #Where everything is saved
if not os.path.exists(outputPath): #Test that outputPath exists
    os.makedirs(outputPath)
    print("Created output directory: ", outputPath)
else:
    print("Output directory is already assigned to: ", outputPath)
```

```
Output directory is already assigned to:  C:\ArcGIS\Projects\Lab3-1
```

Now, I will focus on the elevation data from MNDNR's website first.

I ended up taking a sub-sample so that my computer could process the data better, so I only downloaded four .laz files.

laszip.exe will unzip my .laz files to .las files.

```
#LAZ files

#Variable names
dnrFiles = [ #So named to differentiate from lasFiles
    '4342-28-61.laz', '4342-28-60.laz',
    '4342-29-60.laz', '4342-29-61.laz'
]
laszipUrl = surfaceUrl + 'laszip.exe'
laszipPath = os.path.join(outputPath, 'laszip.exe')

for dnrFile in dnrFiles:
    dnrPath = os.path.join(outputPath, dnrFile)
    if not os.path.exists(dnrPath):
        response = requests.get(surfaceUrl + dnrFile, stream=True)
        if response.status_code == 200:
            with open(dnrPath, 'wb') as file:
                file.write(response.content)
            print(f"Downloaded {dnrFile}")
        else:
            print(f"Failed to download {dnrFile}, status code:
{response.status_code}")
    else:
        print(f"{dnrFile} already exists at {dnrPath}")

#Download laszip.exe if not already done
if not os.path.exists(laszipPath):
    response = requests.get(laszipUrl)
    if response.status_code == 200:
        with open(laszipPath, 'wb') as file:
            file.write(response.content)
        print("Downloaded laszip.exe")
    else:
        print(f"Failed to download laszip.exe, status code:
{response.status_code}")
else:
    print("laszip.exe already downloaded.")

#Ensure laszip.exe is executable
if os.name == 'nt':
    os.chmod(laszipPath, 0o755)
```

```
4342-28-61.laz already exists at C:\ArcGIS\Projects\Lab3-1\4342-28-
61.laz
4342-28-60.laz already exists at C:\ArcGIS\Projects\Lab3-1\4342-28-
60.laz
4342-29-60.laz already exists at C:\ArcGIS\Projects\Lab3-1\4342-29-
60.laz
4342-29-61.laz already exists at C:\ArcGIS\Projects\Lab3-1\4342-29-
61.laz
laszip.exe already downloaded.
```

```
#LAS files & LASD DEM
lasFiles = [os.path.join(outputPath, file.replace('.laz', '.las')) for
file in dnrFiles]
lasdPath = os.path.join(outputPath, 'elevation.lasd')
demPath = os.path.join(outputPath, 'elevation_DEM')

#Unzip .laz files to .las files using laszip
def laszip(dnrPath, lasPath):
    try:
        # Run the laszip command
        result = subprocess.run(
            [laszipPath, '-i', dnrPath, '-o', lasPath],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
            text=True
        )
        # Capture stdout and stderr
        if result.returncode == 0:
            print(f"Successfully unzipped {dnrPath} to {lasPath}")
        else:
            print(f"Error unzipping {dnrPath}:\n{result.stderr}")
    except FileNotFoundError:
        print(f"laszip.exe not found at {laszipPath}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Check if .las files already exist, otherwise unzip
for dnrFile in dnrFiles:
    dnrPath = os.path.join(outputPath, dnrFile)
    lasPath = os.path.join(outputPath, dnrFile.replace('.laz',
'.las'))

    if not os.path.exists(lasPath):
        if os.path.exists(dnrPath):
            laszip(dnrPath, lasPath)
        else:
            print(f"{dnrFile} not found.")
    else:
        print(f"{dnrFile.replace('.laz', '.las')} already exists.")
```

```
4342-28-61.las already exists.
4342-28-60.las already exists.
4342-29-60.las already exists.
4342-29-61.las already exists.
```

In order for me to get this data on my map, I need to make an LAS dataset:

```
#Create LAS Dataset if not already created
if not os.path.exists(lasdPath):
```

```
    arcpy.management.CreateLasDataset(lasFiles, lasdPath)
    print(f"Created LAS dataset: {lasdPath}")
else:
    print(f"LAS dataset already exists at: {lasdPath}")

# Convert LASD to DEM
if not os.path.exists(demPath):
    arcpy.conversion.LasDatasetToRaster(
        lasdPath, demPath,
        value_field="ELEVATION",
        interpolation_type="BINNING AVERAGE NATURAL_NEIGHBOR",
        data_type="FLOAT",
        sampling_type="CELLSIZE",
        sampling_value=2
    )
    print(f"Created DEM: {demPath}")
else:
    print(f"DEM already exists at: {demPath}")
```

```
LAS dataset already exists at: C:\ArcGIS\Projects\Lab3-1\
elevation.lasd
DEM already exists at: C:\ArcGIS\Projects\Lab3-1\elevation_DEM
```

The next section of this part of the lab has to do with the land cover raster I will use to identify water & fields.

Same as before, I need to download and unzip the data.

Then, I am going to clip the data to my project extent, since this data covers the whole of the State of Minnesota.

```
#Land cover raster

#Variable names
zipPath = os.path.join(outputPath, 'landcover.zip')
tifPath = os.path.join(outputPath, 'NLCD_2016_Land_Cover.tif')
clipPath = os.path.join(outputPath, 'land_cover_clip.tif')

#Download land cover raster
if not os.path.exists(zipPath):
    response = requests.get(landcoverUrl)
    if response.status_code == 200:
        with open(zipPath, 'wb') as file:
            file.write(response.content)
        print("Downloaded land cover ZIP file.")
    else:
        print("Failed to download land cover ZIP file, status code: ",
response.status_code)
else:
    print("Land cover ZIP file already exists.")
```

```
#Extract the ZIP file
if not os.path.exists(tifPath):
    with ZipFile(zipPath, 'r') as zip_ref:
        zip_ref.extractall(outputPath)
        print("Extracted all files from the ZIP archive.")
else:
    print("The land cover TIF file already exists.")

#Check if the raster file exists before trying to add it
if os.path.isfile(tifPath):
    # Add the raster to the map
    map.addDataFromPath(tifPath)
    print(f"Added {tifPath} to the map.")
else:
    print("Error: The file NLCD_2016_Land_Cover.tif was not found
after extraction.")

#Clip the new raster to the extent of the DEM
if not os.path.exists(clipPath):
    arcpy.management.Clip(
        in_raster="NLCD_2016_Land_Cover.tif",
        out_raster=r"C:\ArcGIS\Projects\Lab3-1\land_cover_clip.tif",
        in_template_dataset="elevation_DEM",  #Set the extent based on
the DEM layer
        nodata_value="255",
        clipping_geometry="NONE",
        maintain_clipping_extent="NO_MAINTAIN_EXTENT"
    )

#Check that clip worked and delete large .tif file
for layer in map.listLayers():
    if layer.name == "land_cover_clip.tif":
        print("Layer land_cover_clip.tif added to the map.")
    elif layer.name == "NLCD_2016_Land_Cover.tif":
        map.removeLayer(layer)
        print("Layer NLCD_2016_Land_Cover.tif removed from the map.")
```

```
Land cover ZIP file already exists.
The land cover TIF file already exists.
Added C:\ArcGIS\Projects\Lab3-1\NLCD_2016_Land_Cover.tif to the map.
Layer NLCD_2016_Land_Cover.tif removed from the map.
Layer land_cover_clip.tif added to the map.
```

Now I will work on creating the parts of my optimal path analysis. First is a slope raster.

Next, I will make a raster where water & fields are flagged as different than other types of land cover. I did change this from my original code to include wetlands, because otherwise, there would be very little water and it wouldn't be much fun to play with.

Then, I will put the two rasters together to make a cost surface.

```
#Create slope raster for cost surface
slopePath = os.path.join(outputPath, 'slope_DEM.tif')

with arcpy.EnvManager(scratchWorkspace=outputPath):
    slope_DEM = arcpy.sa.Slope(
        in_raster="elevation_DEM",
        output_measurement="DEGREE",
        z_factor=1,
        method="PLANAR",
        z_unit="METER",
        analysis_target_device="GPU_THEN_CPU"
    )
    slope_DEM.save(slopePath)

if arcpy.Exists(slopePath):
    print(f"Slope raster successfully saved as: {slopePath}")
else:
    print("Failed to save the slope raster.")
```

```
Slope raster successfully saved as: C:\ArcGIS\Projects\Lab3-1\
slope_DEM.tif
```

```
#Make raster where water & fields are EQUALLY weighted
equal_land_weight = arcpy.sa.Con(
    (arcpy.sa.Raster(clipPath) == 11) | #Water
    (arcpy.sa.Raster(clipPath) == 90) | #Wetlands
    (arcpy.sa.Raster(clipPath) == 81) | #Pasture
    (arcpy.sa.Raster(clipPath) == 82),  #Crops
    10,
    1
)

weightPath = os.path.join(outputPath, 'equal_land_weight.tif')
equal_land_weight.save(weightPath)

print(f"equal_land_weight saved at: {weightPath}")
```

```
equal_land_weight saved at: C:\ArcGIS\Projects\Lab3-1\
equal_land_weight.tif
```

```
#Make a cost surface where ALL VALUES ARE EQUAL
equalSurface = (arcpy.sa.Raster(slopePath) +
arcpy.sa.Raster(weightPath)) / 2

equalSurfacePath = os.path.join(outputPath, 'equal_surface.tif')
equalSurface.save(equalSurfacePath)

print(f"Cost surface saved at: {equalSurfacePath}")
```

```
Cost surface saved at: C:\ArcGIS\Projects\Lab3-1\equal_surface.tif
```

Finally, I need to make my least cost path.

I had a lot of trouble here until I realized that Esri created a new tool called "Optimal Path As Raster". to replace the least cost path analysis.

The process is the same: define start and end points, create a distance accumulation raster from the start point, and create an optimal path from the end point.

```
#Define start and end points
start = arcpy.Point(568098, 4886440) #easting, northing
startPath = os.path.join(outputPath, 'start.shp')
end = arcpy.Point(570377, 4882902) #Picked a random point in the
Whitewater WMA but not in water
endPath = os.path.join(outputPath, 'end.shp')

#Create the start point feature class if it does not exist
if not arcpy.Exists(startPath):
    arcpy.management.CreateFeatureclass(outputPath, 'start.shp',
'POINT', spatial_reference=ref)
    with arcpy.da.InsertCursor(startPath, ['SHAPE@']) as cursor:
        cursor.insertRow([arcpy.PointGeometry(start, ref)])
    print("Start point feature class created and populated.")
else:
    print("Start point feature class already exists.")

#Create the end point feature class if it does not exist
if not arcpy.Exists(endPath):
    arcpy.management.CreateFeatureclass(outputPath, 'end.shp',
'POINT', spatial_reference=ref)
    with arcpy.da.InsertCursor(endPath, ['SHAPE@']) as cursor:
        cursor.insertRow([arcpy.PointGeometry(end, ref)])
    print("End point feature class created and populated.")
else:
    print("End point feature class already exists.")
```

```
Start point feature class already exists.
End point feature class already exists.
```

```
#Paths to necessary files
distancePath = os.path.join(outputPath, 'cost_distance.tif')
backlinkPath = os.path.join(outputPath, 'backlink.tif')
equalPath = os.path.join(outputPath, 'out_path_all_equal.tif')

#Distance accumulation Raster
cost_distance = arcpy.sa.DistanceAccumulation(
    in_source_data=startPath,
    in_barrier_data=None,
    in_surface_raster=slopePath,  #Use slope_DEM raster
```

```
    in_cost_raster=equalSurfacePath,  #Use equal_surface raster
    in_vertical_raster=None,
    vertical_factor="BINARY 1 -30 30",
    in_horizontal_raster=None,
    horizontal_factor="BINARY 1 45",
    out_back_direction_raster=backlinkPath,  #Save backlink raster
    out_source_direction_raster=None,
    out_source_location_raster=None,
    source_initial_accumulation=None,
    source_maximum_accumulation=None,
    source_cost_multiplier=None,
    source_direction="",
    distance_method="PLANAR"
)

cost_distance.save(distancePath)
print(f"Distance Accumulation raster saved at: {distancePath}")
print(f"Backlink raster saved at: {backlinkPath}")

with arcpy.EnvManager(scratchWorkspace=outputPath):
    out_path_all_equal = arcpy.sa.OptimalPathAsRaster(
        in_destination_data=endPath,
        in_distance_accumulation_raster=distancePath,
        in_back_direction_raster=backlinkPath,
        destination_field="Id",
        path_type="BEST_SINGLE"
    )

out_path_all_equal.save(equalPath)
print(f"Optimal Path raster saved at: {equalPath}")
```

```
Distance Accumulation raster saved at: C:\ArcGIS\Projects\Lab3-1\
cost_distance.tif
Backlink raster saved at: C:\ArcGIS\Projects\Lab3-1\backlink.tif
Optimal Path raster saved at: C:\ArcGIS\Projects\Lab3-1\
out_path_all_equal.tif
```

But that's only the optimal path if Dory weights all her categories the same.

Let's explore what would happen if we gave certain categories different weights.

To start with, Dory is wearing waders. Let's say she really doesn't care all that much about the water; she just want s to stay away from fields and steep slopes.

First, we'll need to make a new raster where water & fields are weighted differently. I'm naming my variables after water since that is the category we are manipulating.

```
#Make a cost surface where avoiding water is not a priority

#Make raster where fields are higher weighted than water
```

```
low_water_weight = arcpy.sa.Con(
    arcpy.sa.Raster(clipPath) == 11, #Water
    5, #Assign low weight
    arcpy.sa.Con(
        arcpy.sa.Raster(clipPath) == 90,  #Wetlands
        5,
        arcpy.sa.Con(
            arcpy.sa.Raster(clipPath) == 81,  #Pasture
            10,  #Assign higher weight
                arcpy.sa.Con(
                    arcpy.sa.Raster(clipPath) == 82,  #Crops
                    10,
                    1  #Default weight for other land cover types
                )
            )
        )
    )

weightPath = os.path.join(outputPath, 'low_water_weight.tif')
low_water_weight.save(weightPath)

print(f"low_water_weight saved at: {weightPath}")

waterSurface = (arcpy.sa.Raster(slopePath) +
arcpy.sa.Raster(weightPath)) / 2

waterSurfacePath = os.path.join(outputPath, 'water_surface.tif')
waterSurface.save(waterSurfacePath)

print(f"Cost surface saved at: {waterSurfacePath}")
```

```
low_water_weight saved at: C:\ArcGIS\Projects\Lab3-1\
low_water_weight.tif
Cost surface saved at: C:\ArcGIS\Projects\Lab3-1\water_surface.tif
```

Okay, now let's make an optimal path using this cost surface where Dory cares less about water than she does about fields:

```
#Paths to necessary files
distancePath = os.path.join(outputPath, 'cost_distance.tif')
backlinkPath = os.path.join(outputPath, 'backlink.tif')
waterPath = os.path.join(outputPath, 'out_path_low_water.tif')

#Distance accumulation Raster
cost_distance = arcpy.sa.DistanceAccumulation(
    in_source_data=startPath,
    in_barrier_data=None,
    in_surface_raster=slopePath,  #Use slope_DEM raster
    in_cost_raster=waterSurfacePath,  #Use water_surface raster
    in_vertical_raster=None,
```

```
    vertical_factor="BINARY 1 -30 30",
    in_horizontal_raster=None,
    horizontal_factor="BINARY 1 45",
    out_back_direction_raster=backlinkPath,  #Save backlink raster
    out_source_direction_raster=None,
    out_source_location_raster=None,
    source_initial_accumulation=None,
    source_maximum_accumulation=None,
    source_cost_multiplier=None,
    source_direction="",
    distance_method="PLANAR"
)

cost_distance.save(distancePath)
print(f"Distance Accumulation raster saved at: {distancePath}")
print(f"Backlink raster saved at: {backlinkPath}")

with arcpy.EnvManager(scratchWorkspace=outputPath):
    out_path_low_water = arcpy.sa.OptimalPathAsRaster(
        in_destination_data=endPath,
        in_distance_accumulation_raster=distancePath,
        in_back_direction_raster=backlinkPath,
        destination_field="Id",
        path_type="BEST_SINGLE"
    )

out_path_low_water.save(waterPath)
print(f"Optimal Path raster saved at: {waterPath}")
```

```
Distance Accumulation raster saved at: C:\ArcGIS\Projects\Lab3-1\
cost_distance.tif
Backlink raster saved at: C:\ArcGIS\Projects\Lab3-1\backlink.tif
Optimal Path raster saved at: C:\ArcGIS\Projects\Lab3-1\
out_path_low_water.tif
```

Now, what if all the fields have fences around them, and Dory cares most about not having to hop fences over avoiding water and steep slopes.

We already have the land cover raster where fields are weighted higher than water bodies, so we can skip that step and move on to the cost surface:

```
#Make a cost surface where avoiding fields is THE priority

fieldSurface = (arcpy.sa.Raster(slopePath) + (2 *
arcpy.sa.Raster(weightPath))) / 3

fieldSurfacePath = os.path.join(outputPath, 'field_surface.tif')
fieldSurface.save(fieldSurfacePath)

print(f"Cost surface saved at: {fieldSurfacePath}")
```

```
Cost surface saved at: C:\ArcGIS\Projects\Lab3-1\field_surface.tif
```

```
#Paths to necessary files
distancePath = os.path.join(outputPath, 'cost_distance.tif')
backlinkPath = os.path.join(outputPath, 'backlink.tif')
fieldPath = os.path.join(outputPath, 'out_path_high_fields.tif')

#Distance accumulation Raster
cost_distance = arcpy.sa.DistanceAccumulation(
    in_source_data=startPath,
    in_barrier_data=None,
    in_surface_raster=slopePath,  #Use slope_DEM raster
    in_cost_raster=fieldSurfacePath,  #Use field_surface raster
    in_vertical_raster=None,
    vertical_factor="BINARY 1 -30 30",
    in_horizontal_raster=None,
    horizontal_factor="BINARY 1 45",
    out_back_direction_raster=backlinkPath,  #Save backlink raster
    out_source_direction_raster=None,
    out_source_location_raster=None,
    source_initial_accumulation=None,
    source_maximum_accumulation=None,
    source_cost_multiplier=None,
    source_direction="",
    distance_method="PLANAR"
)

cost_distance.save(distancePath)
print(f"Distance Accumulation raster saved at: {distancePath}")
print(f"Backlink raster saved at: {backlinkPath}")

with arcpy.EnvManager(scratchWorkspace=outputPath):
    out_path_high_fields = arcpy.sa.OptimalPathAsRaster(
        in_destination_data=endPath,
        in_distance_accumulation_raster=distancePath,
        in_back_direction_raster=backlinkPath,
        destination_field="Id",
        path_type="BEST_SINGLE"
    )

out_path_high_fields.save(fieldPath)
print(f"Optimal Path raster saved at: {fieldPath}")
```

```
Distance Accumulation raster saved at: C:\ArcGIS\Projects\Lab3-1\
cost_distance.tif
Backlink raster saved at: C:\ArcGIS\Projects\Lab3-1\backlink.tif
Optimal Path raster saved at: C:\ArcGIS\Projects\Lab3-1\
out_path_high_fields.tif
```

Okay, now for our third (fourth) optimal path, what if we introduced a new land cover type? Let's say that if there is a road, Dory does NOT want to walk on it, because I don't know, the weatherman said the risk of tractor stampede is high today. With that, Dory also does NOT want to walk down steep hills, and that's going to be weighted higher than anything else.

First, we need a new weighting scheme for land cover:

```
#Make a cost surface where avoiding roads and slopes are priorities

#Make raster that includes roads
road_weight = arcpy.sa.Con(
    arcpy.sa.Raster(clipPath) == 21, #Open asphalt
    10, #Assign high weight
    arcpy.sa.Con(
        arcpy.sa.Raster(clipPath) == 22, #Low intensity road... if I
were being thorough I would include all road types, but this is the
only road type in the study area
        10,
        arcpy.sa.Con(
            arcpy.sa.Raster(clipPath) == 11, #Water
            5, #Assign low weight just cause it makes sense to me
            arcpy.sa.Con(
                arcpy.sa.Raster(clipPath) == 90,  #Wetlands
                5,
                arcpy.sa.Con(
                    arcpy.sa.Raster(clipPath) == 81,  #Pasture
                    10,  #Assign higher weight
                        arcpy.sa.Con(
                        arcpy.sa.Raster(clipPath) == 82,  #Crops
                        10,
                        1  #Default weight for other land cover types
                        )
                    )
                )
            )
        )
    ) #That's a lot of parentheses

weightPath = os.path.join(outputPath, 'road_weight.tif')
road_weight.save(weightPath)

print(f"road_weight saved at: {weightPath}")

roadSurface = ((arcpy.sa.Raster(slopePath) * 3) +
(arcpy.sa.Raster(weightPath) * 2)) / 5 #Where slope is highly weighted

roadSurfacePath = os.path.join(outputPath, 'road_surface.tif')
roadSurface.save(roadSurfacePath)

print(f"Cost surface saved at: {roadSurfacePath}")
```

```
road_weight saved at: C:\ArcGIS\Projects\Lab3-1\road_weight.tif
Cost surface saved at: C:\ArcGIS\Projects\Lab3-1\road_surface.tif
```

```python
#Paths to necessary files
distancePath = os.path.join(outputPath, 'cost_distance.tif')
backlinkPath = os.path.join(outputPath, 'backlink.tif')
roadPath = os.path.join(outputPath, 'out_path_roads_slopes.tif')

#Distance accumulation Raster
cost_distance = arcpy.sa.DistanceAccumulation(
    in_source_data=startPath,
    in_barrier_data=None,
    in_surface_raster=slopePath,  #Use slope_DEM raster
    in_cost_raster=roadSurfacePath,  #Use field_surface raster
    in_vertical_raster=None,
    vertical_factor="BINARY 1 -30 30",
    in_horizontal_raster=None,
    horizontal_factor="BINARY 1 45",
    out_back_direction_raster=backlinkPath,  #Save backlink raster
    out_source_direction_raster=None,
    out_source_location_raster=None,
    source_initial_accumulation=None,
    source_maximum_accumulation=None,
    source_cost_multiplier=None,
    source_direction="",
    distance_method="PLANAR"
)

cost_distance.save(distancePath)
print(f"Distance Accumulation raster saved at: {distancePath}")
print(f"Backlink raster saved at: {backlinkPath}")

with arcpy.EnvManager(scratchWorkspace=outputPath):
    out_path_roads_slopes = arcpy.sa.OptimalPathAsRaster(
        in_destination_data=endPath,
        in_distance_accumulation_raster=distancePath,
        in_back_direction_raster=backlinkPath,
        destination_field="Id",
        path_type="BEST_SINGLE"
    )

out_path_roads_slopes.save(roadPath)
print(f"Optimal Path raster saved at: {roadPath}")
```

```
Distance Accumulation raster saved at: C:\ArcGIS\Projects\Lab3-1\
cost_distance.tif
Backlink raster saved at: C:\ArcGIS\Projects\Lab3-1\backlink.tif
Optimal Path raster saved at: C:\ArcGIS\Projects\Lab3-1\
out_path_roads_slopes.tif
```

That's all, folks!