```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
This script is used to clean up raster files using binary opening
Author: Alexandra Christensen
Created: Wed Oct 14 10:25:58 2020
Updated: 12/01/2022
"""

import rasterio
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
from osgeo import gdal
from pathlib import Path
import networkx as nx
from orinoco import (get_distance_in_channel,
                     get_distance_segments,
                     get_undirected_channel_network,
                     direct_channel_network,
                     export_edges_to_geodataframe,
                     export_nodes_to_geodataframe,
                     get_map_centroid_from_binary_mask,
                     add_flow_attributes,
                     split_tuple_pairs,
                     get_segment_df,
                     get_geo_width_df,
                     update_graph_with_widths,
                     get_array_from_features,
                     get_width_features_from_segments
                     )
from skimage.color import label2rgb
import random
import geopandas as gpd
from shapely.geometry import Point
import geopandas as gpd
import os

import warnings
warnings.filterwarnings('ignore', '.*do not.*', )
warnings.warn('ShapelyDeprecationWarning')
warnings.warn('UserWarning')


def make_distance(water,ocean,folders,delta,pixel_step,xres):

    ocean_mask = ocean.read(1)
    water_mask = np.where((water.read(1)==1) | (ocean_mask==1),1,0)
    profile = ocean.profile
    #ocean_ex = rasterio.open("%s%s_oceanextended.tif" %(temppath,delta))
    #river_ex = rasterio.open("%s%s_riverextended.tif" %(temppath,delta))
    #watermask =
ndimage.binary_closing(array,struct,iterations=3).astype(array.dtype)
```

```
    #ocean_mask =
ndimage.binary_closing(ocean_mask,struct,iterations=c).astype(array.dtype
)

    p1 = profile.copy()
    p1['dtype'] = 'float32'
    p1['COMPRESS'] = 'deflate'

    transform = ocean.transform
    dx, dy = transform.a, -transform.e

    # Build distance raster with the scikit-fmm distance function (phi)
    print('\n######[Make_Channel_Networks][Orinoco -->
get_distance_in_channel] .......\n')
    dist = get_distance_in_channel(water_mask,
                                    ocean_mask,
                                    dx=dx,
                                    dy=dy,
                                    min_rel_area=0) #removes areas with
less than 2.5% of total size

    # mask = np.where(watermask == 0, np.nan,
np.where(ocean_mask==1,np.nan,np.where(np.isnan(dist),0,1)))
    # fillers = _fillnodata(dist,mask,500)

    #dist_clean = np.where(river.read(1)==1,dist,np.nan)
    with rasterio.open("%s%s_distance_%sx%s.tif"
%(folders[8],delta,xres,pixel_step),'w',**p1) as dst:
        dst.write_band(1,dist)

    # Build segment raster according to phi(x)/D where D is threshold
defined by pixel_step * res
    # Connectivity set as 8, edges or corners connectedness
    # Interface adjacent segments are IDs of segments at river/ocean
interface
    # dist = np.where(np.isnan(dist),0,dist)
    print('\n######[Make_Channel_Networks][Orinoco -->
get_distance_segments] .......\n')
    segments, interface_adj_segments = get_distance_segments(dist,
                                                             pixel_step,
                                                             dx=dx,
                                                             dy=dy,

connectivity=8,

min_size=None)
    p2 = profile.copy()
    p2['dtype'] = 'int32'
    p2['COMPRESS'] = 'deflate'
    p2['nodata'] = -9999
    #segments_clean = np.where(river.read(1)==1,segments,0)
    with rasterio.open('%s%s_segments_%sx%s.tif'
%(folders[8],delta,xres,pixel_step), 'w', **p2) as ds:
        ds.write(segments.astype(np.int32), 1)
    if os.path.isfile('%s%s_segments_%sx%s.shp'
%(folders[7],delta,xres,pixel_step)):
        os.remove('%s%s_segments_%sx%s.shp'
%(folders[7],delta,xres,pixel_step))
    ## Polygonize the segments
    print('\n######[Make_Channel_Networks][Orinoco --> segment raster to
shapefile] .......\n')
    os.system('gdal_polygonize.py %s%s_segments_%sx%s.tif
%s%s_segments_%sx%s.shp'
%(folders[8],delta,xres,pixel_step,folders[7],delta,xres,pixel_step))

    segment_interface_slice = np.isin(segments, interface_adj_segments)
    segments_along_interface = segments.copy()
    segments_along_interface[~segment_interface_slice] = 0


    ## Using Region Adjaceny Graph to build network
    ## Nodes = center of segmetns
    ## Edges = network determined by RAG
    ## 8 = allow diagonal connectivity
    print('\n######[Make_Channel_Networks][Orinoco -->
get_undirected_channel_network] .......\n')

    chanG_undirected = get_undirected_channel_network(segments,
                                                  dist,
                                                  profile,
                                                  interface_adj_segments,
                                                  connectivity=8)
    # node_data =dict(chanG_undirected.nodes(data=True))
    ## node key = (x,y) in UTM coordinates
    ## data includes label, meters to interface, x, y, and interface
adjacent (true/false)

    # edge_data =(chanG_undirected.edges(data=True))
    # edge_data = {(e[0], e[1]): e[2] for e in edge_data}
    ## edge key
    ## data includes weight and length_m which are both the straight line
distance between nodes that define the edge


    # fig = plt.subplots(figsize = (50,50))
    # pos = {node: node for node in node_data.keys()}
    # nx.draw(chanG_undirected,
    #     pos=pos,
    #     node_size=1,
    #     node_color='blue')
    # plt.tight_layout()
    # plt.show(block=False)
    # import time
    # time.sleep(5)
    # plt.close('all')
    #
    ## Add direction to the network and prune
    print('\n######[Make_Channel_Networks][Orinoco -->
direct_channel_network] .......\n')
    chanG = direct_channel_network(chanG_undirected,
                                    # The keywords below are how the
pruning occurs
                                    # Specifies pruning will be done
                                    remove_dangling=True,
                                    # Do not prune nodes within 1 km of
interface
                                    interface_buffer_m=1_0,
                                    # Remove edge groups with an degree 1
endpoint and size <=3

group_min_size=3,#xres*pixel_step,#*100,
                                    # How many times to do this pruning
                                    dangling_iterations=1
                                    )

    fig = plt.subplots(figsize = (50,50))
    node_data =dict(chanG.nodes(data=True))
    edge_data =(chanG.edges(data=True))
    edge_data = {(e[0], e[1]): e[2] for e in edge_data}
    pos = {node: node for node in node_data.keys()}
    nx.draw(chanG,
        pos=pos,
        node_size=1,
        node_color='blue')
    plt.tight_layout()
    plt.savefig("%s%s_centerline.png" %(folders[9],delta))
    plt.close()

    # ocean_centroid = get_map_centroid_from_binary_mask(ocean_mask,
profile)
    # df_ocean_centroid =
gpd.GeoDataFrame(geometry=[Point(ocean_centroid)], crs='EPSG:4326')
    # connected_to_interface = [node for node in chanG.nodes() if
(chanG.nodes[node]['interface_adj'])]
    # chanG_sink = chanG.copy()
    # edge_data_to_interface = {(node, ocean_centroid): {'weight':
0,'meters_to_interface': 0} for node in connected_to_interface}
    # chanG_sink.add_edges_from(edge_data_to_interface.keys())
    # #pos[ocean_centroid] = ocean_centroid
    ocean_centroid = get_map_centroid_from_binary_mask(ocean_mask,
profile)
    df_ocean_centroid =
gpd.GeoDataFrame(geometry=[Point(ocean_centroid)], crs='EPSG:4326')

    connected_to_interface = [node for node in chanG.nodes() if
(chanG.nodes[node]['interface_adj'])]
    chanG_sink = chanG.copy()

    edge_data_to_interface = {(node, ocean_centroid): {'weight':
0,'meters_to_interface': 0} for node in connected_to_interface}
```

```
    chanG_sink.add_edges_from(edge_data_to_interface.keys())
    #pos[ocean_centroid] = ocean_centroid


    print('\n######[Make_Channel_Networks][Orinoco -->
add_flow_attributes] .......\n')
    chanG = add_flow_attributes(chanG, dist, profile['transform'])
    # node_data =dict(chanG.nodes(data=True))
    # random.choice(list(node_data.items()))

    df_segments = get_segment_df(segments, chanG, profile,8)

    print('\n######[Make_Channel_Networks][Orinoco --> get_geo_width_df]
.......\n')
    df_geo_widths = get_geo_width_df(df_segments,
                                      chanG,
                                      # How many hops to permit in our
neighborhood
                                      radius=1)
    df_geo_widths_out = split_tuple_pairs(df_geo_widths)
    #df_geo_widths_out.to_file('%s%s_width_segments_%s.shp'
%(folders[6],delta,round(pixel_step*xres)), driver='ESRI Shapefile')

    print('\n######[Make_Channel_Networks][Orinoco -->
update_graph_with_widths] .......\n')
    chanG = update_graph_with_widths(chanG, df_geo_widths)

    print('\n######[Make_Channel_Networks][Orinoco -->
export_edges_to_geodataframe] .......\n')
    df_edges = export_edges_to_geodataframe(chanG, profile['crs'])
    df_edges.rename(columns={'edges_in_segment':'edgesinseg'},
inplace=True)
    # try:
    #     os.remove('%s%s_river_centerline_%sx%s.shp'
%(folders[7],delta,xres,pixel_step))
    # except: ''
    df_edges.to_file('%s%s_river_centerline_%sx%s.shp'
%(folders[7],delta,xres,pixel_step), driver='ESRI Shapefile')
    df_edges2 = df_edges.to_crs(4326)
    df_edges2.to_file('%s%s_river_centerline_%sx%s.geojson'
%(folders[7],delta,xres,pixel_step), driver='GeoJSON')

    print('\n######[Make_Channel_Networks][Orinoco -->
export_nodes_to_geodataframe] .......\n')
    df_nodes = export_nodes_to_geodataframe(chanG, profile['crs'])
    df_nodes = df_nodes.drop(columns ='flow_vector_perp_grad')
    df_nodes2 = split_tuple_pairs(df_nodes)
    df_nodes2 = df_nodes2.to_crs(4326)

    df_nodes2.to_file('%s%s_river_centernode_%sx%s.geojson'
%(folders[7],delta,xres,pixel_step), driver='GeoJSON')

df_nodes.rename(columns={'meters_to_interface':'m_to_inter','interface_ad
j':'inter_adj','graph_degree':'graph_deg','flow_vector_network':'net','fl
ow_vector_perp':'perp' ,'flow_vector_perp_network':
'perpnet','flow_vector_perp_grad':'perpgrad'}, inplace=True)
    df_nodes = split_tuple_pairs(df_nodes)
    df_nodes.to_file('%s%s_river_centernode_%sx%s.shp'
%(folders[7],delta,xres,pixel_step), driver='ESRI Shapefile')

    print('\n######[Make_Channel_Networks][Orinoco -->
get_width_features_from_segments] .......\n')
    width_features = get_width_features_from_segments(segments,profile)
    print('\n######[Make_Channel_Networks][Orinoco -->
get_array_from_features] .......\n')
    widths = get_array_from_features(segments,width_features)
    #widths2 = np.where(river.read(1)==1,widths,np.nan)

    p = profile.copy()
    p['dtype'] = 'float32'
    p['COMPRESS'] = 'deflate'
    #fillers = _fillnodata(widths,mask,500)
    #widths =
np.where(watermask==0,np.nan,np.where(np.isnan(widths),fillers,widths))
    with rasterio.open("%s%s_widths_%sx%s.tif"
%(folders[8],delta,xres,pixel_step), 'w', **p) as ds:
        ds.write(widths.astype(np.float32), 1)


def opening(array, a,b,c):
    struct = np.ones([a,b])
    newarray =
ndimage.binary_opening(array,struct,iterations=c).astype(array.dtype)
    newarray[newarray==1] = 2
    newarray[newarray==0] = 1
    newarray[newarray==2] = 0
    return newarray
def closing(array,a,b,c):
    struct = np.ones([a,b])
    newarray =
ndimage.binary_closing(array,struct,iterations=c).astype(array.dtype)
    newarray[newarray==1] = 2
    newarray[newarray==0] = 1
    newarray[newarray==2] = 0
    return newarray
```