# Shortest path in a bidimensional space using visgraph library

En aquest notebook utilitzarem la tècnica dels grafs de visibilitat per a trobar el camí més curt entre dos punts en un espai bidimensional donats una sèrie d'obstacles definits en forma de polígon.

Tot i que la versió original de `pyvisgraph` (la llibreria usada), només implementa l'algorisme de `Dijkstra` per tal de trobar el camí més curt entre dos punts en un graf, nosaltres usarem una versió modificada que inclou l'algorisme `A*`, per comparar el seu rendiment.

Hem partit dels exemples existents a la llibreria per tal de realitzar aquest notebook, tot i que amb algunes modificacions.

Comencem carregant les llibreries necessàries:

```
import geopandas as gpd
import geoviews as gv
import geoviews.feature as gf
import hvplot.pandas
import shapefile
import pyvisgraph as vg
import holoviews as hv
from holoviews import opts
from pyvisgraph.visible_vertices import visible_vertices
import time
import matplotlib.pyplot as plt
from PIL import Image
from IPython.display import display_png

renderer = hv.renderer('bokeh')
```

El primer exemple consisteix en definir dos obstacles simples, per després poder trobar el camí més curt donats dos punts.

Així doncs, primer comencem definint aquests obstacles:

```
polys = [[vg.Point(0.0,1.0), vg.Point(3.0,1.0), vg.Point(1.5,4.0)],
         [vg.Point(4.0,4.0), vg.Point(7.0,4.0), vg.Point(5.5,8.0)]]
```

I en construïm el seu graf de visibilitat:

```
g = vg.VisGraph()
g.build(polys)
```

```
100%|
 1/1 [00:00<?, ?it/s]
```

A partir d'aquí, podem calcular el camí més curt entre dos punts donats:

```
shortest = g.shortest_path(vg.Point(1.5,0.0), vg.Point(8.0, 8.0))
shortest
```

```
[Point(1.50, 0.00), Point(3.00, 1.00), Point(7.00, 4.00), Point(8.00, 8.00)]
```

La sèrie de passos següents són necessaris per tal de representar gràficament el graf de visibilitat i el camí més curt obtinguts.

Primer, canviem el format dels punts del camí més curt (ho convertim a llista de tuples d'enters):

```
shortest_path = [(point.x, point.y) for point in shortest]
```

Fem el mateix amb els punts dels polígons. Fins ara estàvem representant punts en l'espai, però ara ens interessa construir un camí que uneixi aquests punts. Per això, dupliquem el punt inicial (per 'tancar la línia').

```
paths = []
for poly in polys:
    poly_aux = []
    for point in poly:
        poly_aux.append((point.x, point.y))
    poly_aux.append(poly_aux[0])
    paths.append(poly_aux)
paths
```

```
[[(0.0, 1.0), (3.0, 1.0), (1.5, 4.0), (0.0, 1.0)],
 [(4.0, 4.0), (7.0, 4.0), (5.5, 8.0), (4.0, 4.0)]]
```

Els ajuntem en una sola llista:

```
all_points = set([point for point in poly for poly in polys] + shortest)
all_points
```

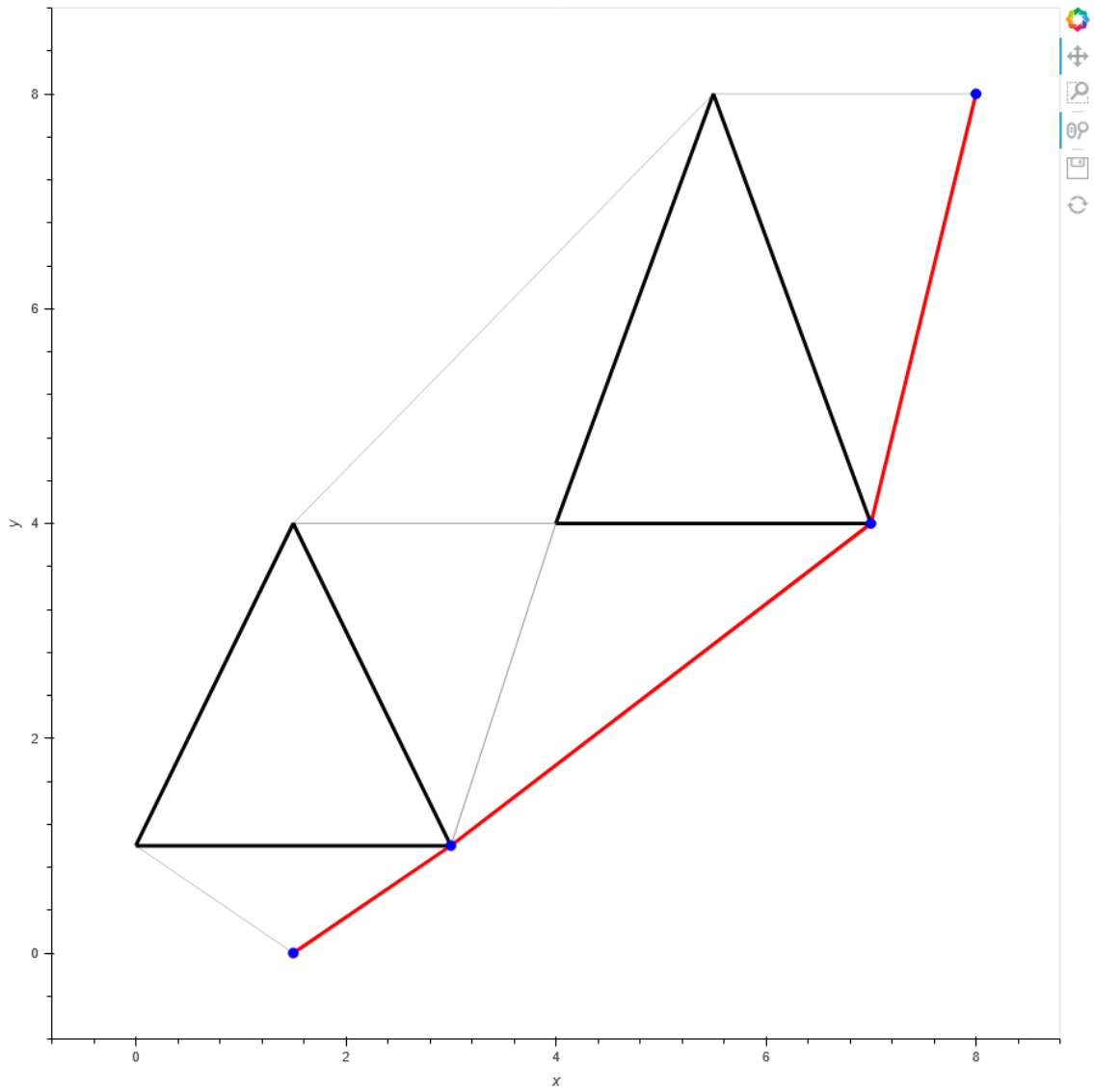
```
{Point(4.00, 4.00),
 Point(8.00, 8.00),
 Point(3.00, 1.00),
 Point(7.00, 4.00),
 Point(1.50, 0.00),
 Point(5.50, 8.00)}
```

I calculem, per cada punt del mapa, quins són els punts visibles:

```
visible_points = []

for start_point in all_points:
    visible = visible_vertices(start_point, g.graph, None, None)
    for point in visible:
        visible_points.append([(start_point.x, start_point.y), (point.x, point.y)])
```

Finalment, representem els resultats obtinguts del graf de visibilitat. En blau, els vèrtexs del camí mínim, i les arestes en vermell. En color negre, les arestes del graf que representen els obstacles, i la resta d'arestes del graf, en gris.

```
overlay = hv.Path(visible_points).opts(opts.Path(color='grey', line_width=0.5)) \
    * hv.Path(paths).opts(opts.Path(color='black', line_width=3)) \
        * hv.Path(shortest_path).opts(opts.Path(color='red', line_width=3)) \
            * hv.Points(shortest_path).opts(opts.Points(color='blue', size=8))

plot = overlay.opts(width=900, height=900)

png, info = renderer(plot, fmt='png')
display_png(png, raw=True)
```

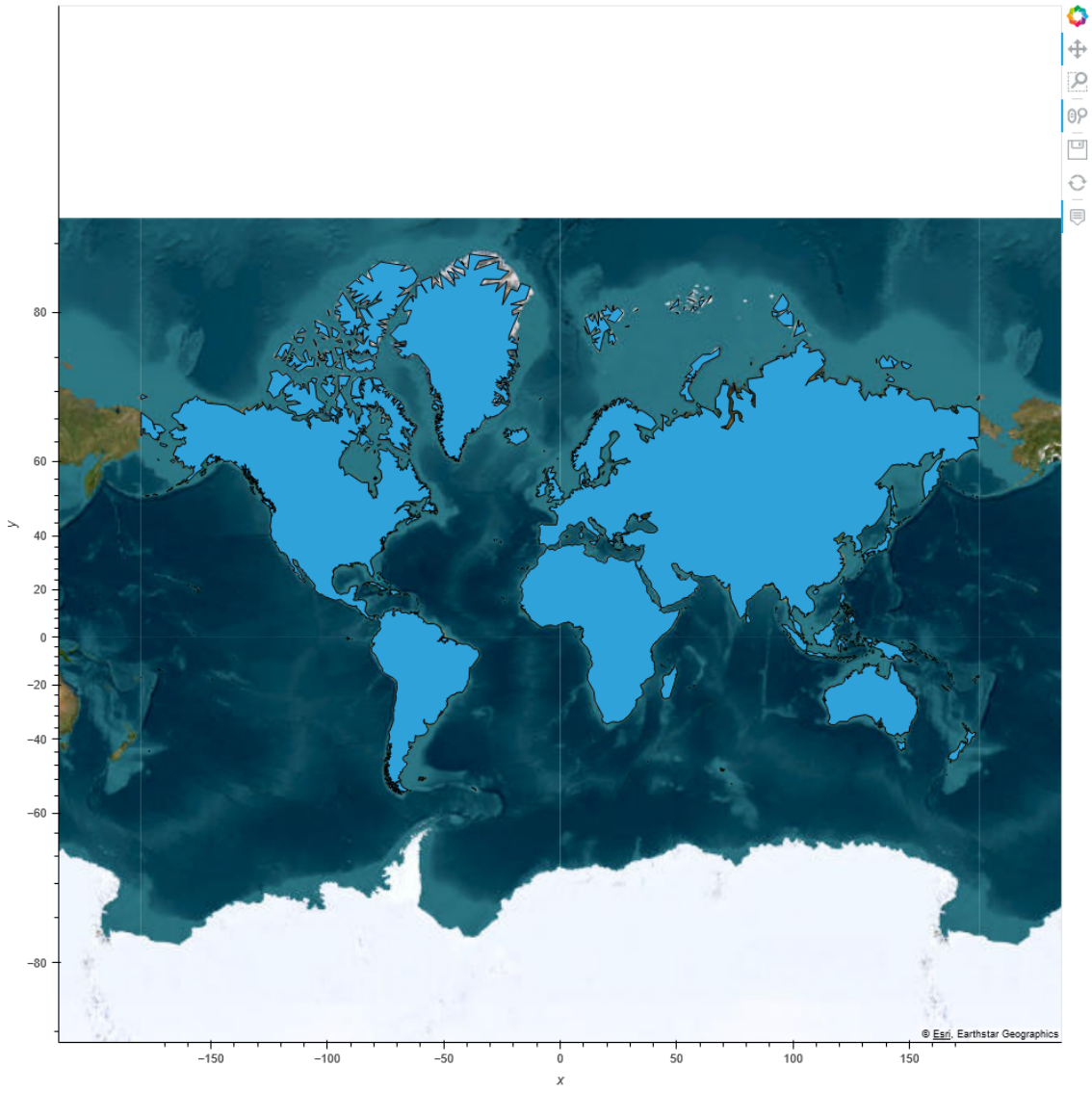



El segon exemple és semblant al primer, però trobant el camí més curt per mar donats dues coordenades qualsevols de la superficie terrestre. El primer que hem de fer és importar el fitxer `shoreline/GSHHS_c_L1`. Aquest fitxer és de tipus `shapefile`, i conté una capa poligonal dels continents (els obstacles):

```
input_shapefile = shapefile.Reader('shoreline/GSHHS_c_L1')
```

Ara el tornem a llegir però amb la llibreria `geopandas`, per tal de poder representar-lo gràficament:

```
df = gpd.read_file('shoreline/GSHHS_c_L1.dbf')
plot = df.hvplot(width=1000, height=1000, geo=True, tiles='ESRI')

png, info = renderer(plot, fmt='png')
display_png(png, raw=True)
```




Mirem quants polígons diferents hi ha:

```
shapes = input_shapefile.shapes()
print('El shapefile conté {} polígons.'.format(len(shapes)))
```

```
El shapefile conté 742 polígons.
```

I també quants punts diferents formen aquests polígons:

```
points = sum([len(shape.points) for shape in shapes])

print(f"El shapefile conté {points} punts.")
```

```
El shapefile conté 7282 punts.
```

Construïm el graf de visibilitat executant la següent cel·la de codi. És un procés costós, que tarda uns 15 minuts. Si el paral·lelitzem, en el nostre cas en `10` nuclis, aleshores tarda uns segons. També el guardem a disc, per després poder-lo recuperar si ens interessa.

```
# Crear llista de polígons
polygons = []
for shape in shapes:
    polygon = []
    for point in shape.points:
        polygon.append(vg.Point(point[0], point[1]))
    polygons.append(polygon)

# Construïr el graf de visibilitat
output_graphfile = 'GSHHS_c_L1.graph'
graph = vg.VisGraph()
graph.build(polygons, workers = 10)

graph.save(output_graphfile)
```

```
100%|
 654/654 [02:29<00:00,  4.37it/s]
```

Ara passem a definir dos punts qualssevol (en coordenades GPS), que representaran el punt inicial i final del camí:

```
start_point = vg.Point(12.568337, 55.676098) # Copenhagen
end_point = vg.Point(103.851959, 1.290270) # Singapur
```

En calculem el camí més curt, primer utilitzant l'algorisme de `Dijkstra`:

```
startTime = time.time()
shortest_path_dijsktra  = graph.shortest_path(start_point, end_point)
print("Camí més curt amb algorisme de Dijkstra's trobat en {}
segons.".format(time.time() - startTime))
```

```
Camí més curt amb algorisme de Dijkstra's trobat en 0.9575605392456055 segons.
```

I altra vegada, però utilizant l'algorisme l'algorisme de `A*`:

```
startTime = time.time()
shortest_path_astar  = graph.shortest_path(start_point, end_point, solver = "astar")
print("Camí més curt amb algorisme de A*'s trobat en {} segons.".format(time.time() -
startTime))
```

```
Camí més curt amb algorisme de A*'s trobat en 0.5655708312988281 segons.
```

Veiem que el **temps d'execució és considerablement menor** (la meitat aproximadament).

Ara, tornem a fer un canvi de format per poder representar gràficament la solució:

```
visible_points = []

for start_point in shortest_path_astar:
    visible = visible_vertices(start_point, graph.graph, None, None)
    for point in visible:
        visible_points.append([(start_point.x, start_point.y), (point.x, point.y)])
```

I representem. Enlloc de tot el graf, simplement mostrem el camí més curt, i les arestes que porten als punts visibles des dels punts del camí més curt (ja que altrament tindríem massa arestes):

```
points = [(point.x, point.y) for point in shortest_path_dijsktra]
path = gv.Path(points)

# Customize the appearance of the path and add a basemap
plot = df.hvplot(width=1500, height=1000, geo=True, tiles='ESRI') \
    * gv.Path(visible_points).opts(opts.Path(color='grey', line_width=1)) \
        * path.opts(opts.Path(color='red', line_width=3)) \
            * gv.Points(path).opts(opts.Points(color='k', size=8))

png, info = renderer(plot, fmt='png')
display_png(png, raw=True)
```