COMPONENTES CONEXAS:

PRÁCTICA VOLUNTARIA

Autor: Adrián Pérez Peinador

Introducción/motivación

El principal objetivo de esta práctica es diseñar un algoritmo para calcular el número de componentes conexas de la unión de n segmentos, A. Tras el diseño de este algoritmo se pide usarlo en un caso dado con $n = 1000$ y después se pide encontrar el mínimo número de segmentos necesarios para conectar el conjunto A. Debido al elevado coste computacional de esta tarea, el objetivo será diseñar otro algoritmo para encontrar en un tiempo razonable un número suficientemente pequeño de segmentos que conecten A.

Material usado

En el código se hace uso de varias librerías de Python. En concreto se usan numpy (para los arrays de numpy), matplotlib (a la hora de representar A), random (para la generación de números aleatorios) y time (para medir el tiempo de ejecución). Por otro lado se hace uso del código que genera A, proporcionado por el profesor.

Resultados

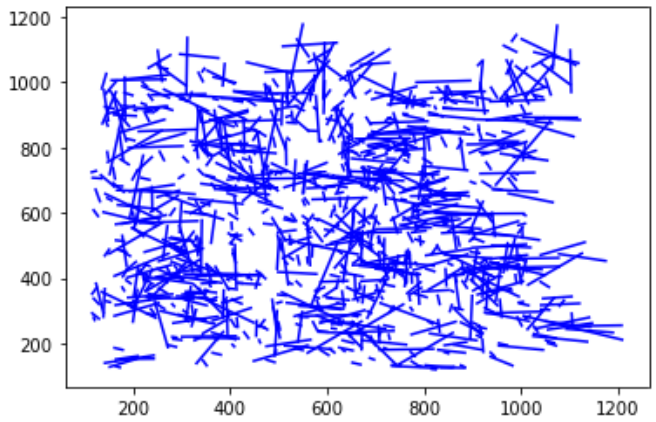
A continuación se exponen los resultados obtenidos para cada apartado pedido, así como una idea de la forma de obtenerlos.

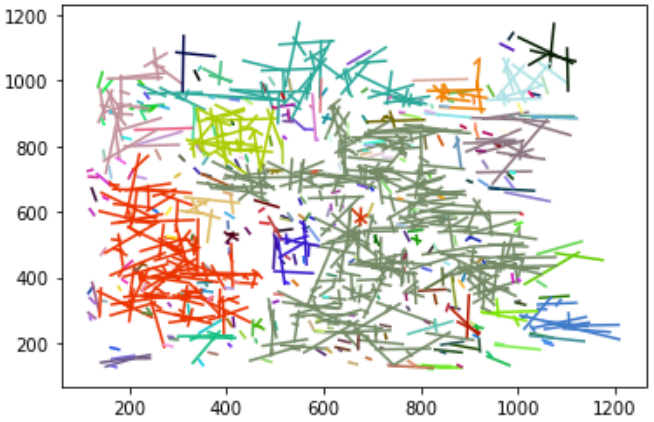
Apartado 1:

En este apartado se pide el diseño e implementación del algoritmo que calcula el número de componentes conexas. Para este algoritmo se usa la función *intersecan*, que dados dos segmentos indica si se intersecan. El funcionamiento del algoritmo comienza inicializando cada segmento como una componente conexa. Después, por cada segmento se comprueba si interseca con cada uno de los demás y en caso afirmativo se unen sus componentes conexas. El resultado es un vector que indica la componente a la que pertenece cada segmento. La función *componentesConexas* implementa este algoritmo en Python y representa gráficamente cada componente conexa de un color.

Apartado 2:

En este apartado se calcula el número de componentes conexas de A usando la función *componentesConexas* del apartado 1, obteniéndose las **338 componentes conexas** que se representan en diferentes colores en la segunda figura (la primera representa simplemente A).





Además, en este apartado se pide hallar el mínimo número m de segmentos que conectan A. Para ello, usaremos que dados dos segmentos no paralelos, estos se pueden alargar hasta cortarse. Por tanto se puede acotar $m \leq (n+1)/2$, pues siempre se puede encontrar un segmento que corte a dos componentes conexas, y se pueden elegir que corten a las n y no sean paralelos.

Usando esto, basta encontrar m segmentos que corten entre todos a todas las componentes, asumiendo que no todos serán paralelos. Haremos una búsqueda voraz de segmentos que corten al mayor número posible de componentes no cortadas anteriormente. Para ello usamos el parámetro *minc* (mínimo número de componentes nuevas que debe cortar un segmento para aceptarlo), y generaremos aleatoriamente segmentos que atraviesen el cuadrado $[0, 1250]^2$ y corten al menos a *minc* componentes nuevas. Cuando pasen T segundos sin encontrar segmentos válidos decrementaremos *minc* en uno y seguiremos hasta haber encontrado segmentos que corten a todas las componentes. La función *segmentosNuevos* implementa este algoritmo para unos parámetros *minc* y T dados, y al ejecutarla varias veces se han obtenido los siguientes datos experimentales:

- El menor número de segmentos encontrados que conectan A ha sido **49 segmentos.**
- Unos valores buenos para los parámetros *minc* y T son 10 y 5 respectivamente, aunque *minc* se puede subir hasta 20.
- Los resultados obtenidos con estos valores no se mejoran sustancialmente aumentandolos.

Conclusión

A la vista de los resultados obtenidos, podemos concluir por un lado que la función *componentesConexas* encuentra el número de componentes conexas de la unión de n segmentos de forma rápida y eficiente para las características del problema. Por otro lado, podemos decir que la función *segmentosNuevos* encuentra un número de segmentos que conectan A suficientemente pequeño en un tiempo razonable. Los resultados de 49 segmentos mejoran sustancialmente la cota teórica de $(n+1)/2$, que en este caso era de 169. De cualquier manera, se pueden modificar los parámetros *minc* y T para encontrar soluciones mejores. Sin embargo, siguiendo la regla de Pareto, para mejorar un 20% los resultados es necesario hacer un 80% del trabajo.

Código

A continuación se anexa el código escrito para esta práctica, comentado para aclarar al lector lo que se hace en cada momento.

Código provisto por el profesor para generar A:

```
# -*- coding: utf-8 -*-
import random
import numpy as np
import matplotlib.pyplot as plt
import time

################################ PARTE 1 ###################################

#Generamos 1000 segmentos aleatorios, pero siempre serán los mismos

#Usaremos primero el concepto de coordenadas
X = []
Y = []

#Fijamos el modo aleatorio con una versión prefijada. NO MODIFICAR!!
random.seed(a=1, version=2)

#Generamos subconjuntos cuadrados del plano R2 para determinar los rangos de X e Y
xrango1 = random.sample(range(100, 1000), 200)
xrango2 = list(np.add(xrango1, random.sample(range(10, 230), 200)))
yrango1 = random.sample(range(100, 950), 200)
yrango2 = list(np.add(yrango1, random.sample(range(10, 275), 200)))

for j in range(len(xrango1)):
    for i in range(5):
        random.seed(a=i, version=2)
        xrandomlist = random.sample(range(xrango1[j], xrango2[j]), 4)
        yrandomlist = random.sample(range(yrango1[j], yrango2[j]), 4)
        X.append(xrandomlist[0:2])
        Y.append(yrandomlist[2:4])

#Representamos el Espacio topológico representado por los 1000 segmentos

for i in range(len(X)):
    plt.plot(X[i], Y[i], 'b')
plt.show()

############################################################################
```

Funciones componentesConexas e intersecan:

```
def intersecan(x1, y1, x2, y2):
    a1 = x1[1] - x1[0]
    b1 = x2[0] - x2[1]
    c1 = x2[0] - x1[0]
    a2 = y1[1] - y1[0]
    b2 = y2[0] - y2[1]
    c2 = y2[0] - y1[0]
    d = a1*b2-a2*b1
    if d == 0:  # Si son paralelos
        if a1*c2 != a2*c1:  # Si no están en la misma recta
            return False
        if x1[0] == x1[1]:
            if min(y1) > max(y2) or min(y2) > max(y1):
                return False
        else:
            if min(x1) > max(x2) or min(x2) > max(x1):
                return False
        return True
    t = (c1*b2-c2*b1)/d
    s = (a1*c2-a2*c1)/d
    return t >= 0 and t <= 1 and s >= 0 and s <= 1


def componentesConexas(_X, _Y):
    n = len(_X)
    comp = np.arange(n)
    for i in range(n):
        cambiado = False
        for j in range(i):
            if comp[i] != comp[j] and intersecan(_X[i], _Y[i], _X[j], _Y[j]):
                if cambiado:
                    for k in range(i+1):
                        if comp[k] == comp[i]:
                            comp[k] = comp[j]
                else:
                    comp[i] = comp[j]
                    cambiado = True

    for i in range(n):
        r = comp[i]/1000
        g = (comp[i] % 100)/100
        b = (comp[i] % 10) / 10
        plt.plot(X[i], Y[i], color = (r,g,b))
    plt.show()

    return comp


comp = componentesConexas(X, Y)
print("Número de componentes:", len(set(comp)))
```

Función segmentosNuevos:

```python
def segmentosNuevos(minc, T, X, Y, comp):
    n = len(X)
    conectadas = set()
    nuevos = 0
    t = time.time()
    N = len(set(comp))

    while len(conectadas) < N:
        l1 = random.randint(0,3)
        l2 = l1
        while l1 == l2:
            l2 = random.randint(0,3)
        a1 = random.randint(0, 1250)
        a2 = random.randint(0, 1250)

        if l1 == 0:
            x1, y1 = 0, a1
        elif l1 == 1:
            x1, y1 = a1, 0
        elif l1 == 2:
            x1, y1 = a1, 1250
        else:
            x1, y1 = 1250, a1

        if l2 == 0:
            x2, y2 = 0, a2
        elif l2 == 1:
            x2, y2 = a2, 0
        elif l2 == 2:
            x2, y2 = a2, 1250
        else:
            x2, y2 = 1250, a2

        conecta = 0
        aux = []
        for i in range(n):
            if comp[i] not in conectadas and comp[i] not in aux
                    and intersecan(X[i], Y[i], [x1,x2], [y1,y2]):
                conecta += 1
                if conecta < minc:
                    aux.append(comp[i])
                else:
                    conectadas.add(comp[i])
                    for c in aux:
                        conectadas.add(c)
        if conecta >= minc:
            t = time.time()
            nuevos += 1
            #print(len(conectadas))
        elif time.time() - t > T:
            if minc > 2:
                t = time.time()
                minc -= 1
            elif N - len(conectadas) <= 2:
                nuevos += 1
                conectadas.add(-1)
                conectadas.add(-2)
    return nuevos
```