Taller 8

Luis Alfredo Acosta Correales
C.C. 1065562875

José Orlando Maldonado Bautista
Docente

Computación Paralela
Programa Ingeniería De Sistemas
Facultad De Ingenierías Y Arquitectura
Universidad De Pamplona
2022

Taller 8

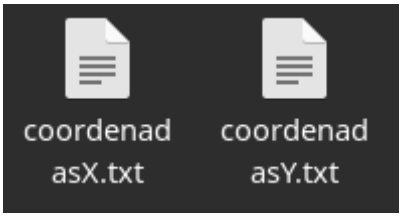
Procesar ficheros

Recibe el archivo en formato tsp, lo procesa y crea dos archivos txt

CoordenadasX.txt

CoordenadasY.txt


```python
def readFile(nombre):
    with open(nombre, 'r') as tsp:
        file =  tsp.readlines()
        positionDisplayDataSection = file.index('NODE_COORD_SECTION\n')
        displayDataSection = file[positionDisplayDataSection+1:]
        return displayDataSection;

def createVCordenadas(coordenadas):
    coordenadasX = []
    coordenadasY = []
    for elem in coordenadas[:len(coordenadas)-1]:
        coordenada = elem.split(' ')[1:]
        newCoordenada = [x for x in coordenada if x != '']
        coordenadasX.append(newCoordenada[1])
        coordenadasY.append(newCoordenada[2].split('\n')[0])
    return [coordenadasX, coordenadasY]

def writeFile(nombre, vector):
    f = open(nombre, mode="w")
    for i in vector:
        f.write(f"{i}\n")
    f.close()

if __name__ == "__main__":
    coordenadas = readFile("att48.tsp")
    [coorX, coorY] = createVCordenadas(coordenadas)
    writeFile("coordenadasX.txt", coorX)
    writeFile("coordenadasY.txt", coorY)
```



- Se leen los archivos creados anteriormente y se guardan los valores en los arreglos del algoritmo genético

```
void createCoordinateVectors(int Coordinates[]){
    ifstream file;
    string linea;
    file.open(COORDINATES_X, ios::in);
    int c=0;
    while (!file.eof())
    {
        getline(file,linea);
        if (linea != "")
        {
            Coordinates[c]=std::stod(linea);
            c++;
        }
    }
}
```

```
createCoordinateVectors(coordenadasX);
createCoordinateVectors(coordenadasY);
```

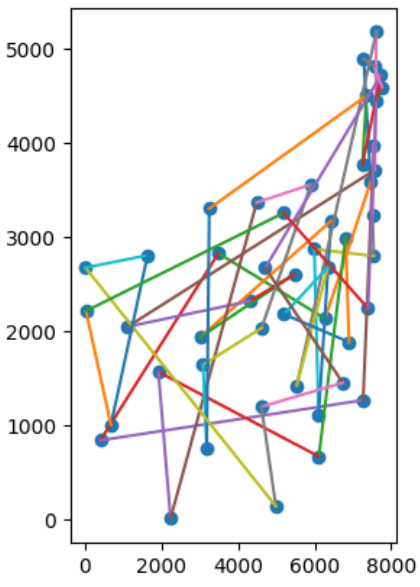
- Se declaran los parámetros del algoritmo

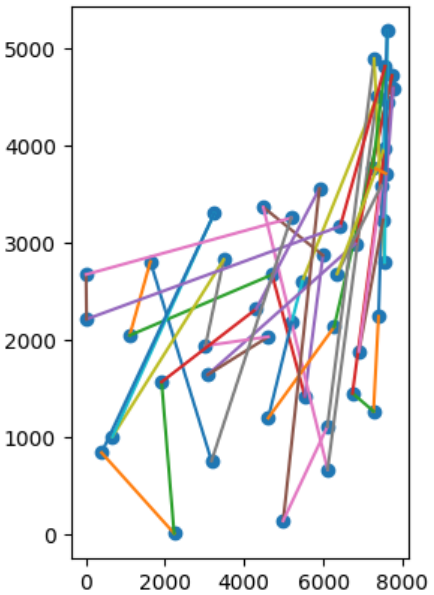
```
int nCiudades = DIM1; // nodos del grafo
int tamPoblacion = 40000; // tama□o de la poblaci□n
int posMejor = 0; // posici□n del individuo mejor adaptado
int numMaxGen = 2500;
```

- Ejecución del algoritmo genético simple

```
======Fundamentos de computaci▒n paralela y distribuida ======
      Algoritmo Gen▒tico Simple para el problema del TSP
                Versi▒n serial y paralela

 N▒mero de ciudades: 48
 Tama▒o de la poblaci▒n: 1000
 Numero de generaciones: 250

Poblaci▒n inicial :
camino :

 Longitud del camino : 112458.27

 ========= INICIO DEL ALGORITMO GENETICO ==========

 ---- alcanza a pasar---

 ---- llego aqui---

 Longitud del camino final : 91097.98
```

- Ejecución del archivo GraficarTSP.py





Modelo de isla (Anillo)

- Todas las funciones necesarias se guardaron en un archivo utils.cpp, que posteriormente, fueron importadas en el archivo modeloIslas.cpp para poder hacer uso de estas
- Se añadieron dos funciones más para poder hacer el modelo de isla
- Función aplicar intercambios donde recibe dos islas, extrae los mejores individuos de la isla a y de la isla b para luego insertarlos de forma cruzada

```
void aplicarIntercambio(int isla_a[],int isla_b[],int posMejor_a,
                        int posMejor_b, int nCiudades,int tamSubPoblacion) {
    int *sujeto_a,*sujeto_b;
    sujeto_a = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    sujeto_b = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    extraerVector(isla_a,sujeto_a,posMejor_a,tamSubPoblacion,nCiudades);
    extraerVector(isla_b,sujeto_b,posMejor_b,tamSubPoblacion,nCiudades);
    insertarVector(isla_a,sujeto_b,posMejor_a,tamSubPoblacion,nCiudades);
    insertarVector(isla_b,sujeto_a,posMejor_b,tamSubPoblacion,nCiudades);
}
```

- Dividir población, esta función divide la población en 4 partes iguales y cada parte extraída la inserta en una isla

```
void dividirPoblacion(int tamPoblacion,int poblacion[],int isla1[],
                      int isla2[],int isla3[],int isla4[],int nCiudades) {
    int *sujeto,n,x;
    n=tamPoblacion/4;
    sujeto = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    x=0;
    for(int i=0; i<tamPoblacion; i++) {
        if(i<n) {//rango de datos para la Isla 1
            extraerVector(poblacion,sujeto,i,tamPoblacion,nCiudades);
            insertarVector(isla1,sujeto,x,n,nCiudades);
            x++;
        }
        if(i>=n&&i<n*2) {//rango de datos para la isla 2
            if(i==n) {
                x=0;
            }
            extraerVector(poblacion,sujeto,i,tamPoblacion,nCiudades);
            insertarVector(isla2,sujeto,x,n,nCiudades);
            x++;
        }
        if(i>=n*2&&i<n*3) {//rango de datos para la Isla 3
            if(i==n*2) {
                x=0;
            }
            extraerVector(poblacion,sujeto,i,tamPoblacion,nCiudades);
            insertarVector(isla3,sujeto,x,n,nCiudades);
            x++;
        }
        if(i>=n*3&&i<n*4) {// rango de datos paraIsla la 4
            if(i==n*3) {
                x=0;
            }
            extraerVector(poblacion,sujeto,i,tamPoblacion,nCiudades);
            insertarVector(isla4,sujeto,x,n,nCiudades);
            x++;
        }
    }
}
```

- Se declaran las variables y punteros necesarios para la ejecución en el main

```c
int main() {

    // Semilla para n◆meros aleatorios
    srand(time(NULL));

    // Declaracion variables - par◆metros del algoritmo
    int nCiudades=DIM1; // nodos del grafo
    int tamPoblacion = 500; // tama◆o de la poblaci◆n
    int posMejor_1 = 0,posMejor_2 = 0,posMejor_3 = 0,posMejor_4 = 0; // posici◆nes de los individuos mejores adaptados de cada isla
    int numMaxGen = 1000;
    int mutados=0;

    // Variables dinamicas para almacenamiento de vectores y matrices
    int *poblacion, *padre1, *padre2, *hijo1, *hijo2,*isla1,*isla2,*isla3,*isla4;
    double  *coordenadasX, *coordenadasY,*matrizDistancias,*distancias,*aptitud;

```

```c
//Punteros correspondietes a cada isla
    double *distancias_1, *aptitud_1, *puntuacion_1,
           *puntAcumulada_1, *nuevaAptitud_1, *nuevaDistancias_1;
    int *mejorCamino_1, *pobAuxiliar_1;

    double *distancias_2, *aptitud_2, *puntuacion_2,
           *puntAcumulada_2, *nuevaAptitud_2, *nuevaDistancias_2;
    int *mejorCamino_2, *pobAuxiliar_2;

    double *distancias_3, *aptitud_3, *puntuacion_3,
           *puntAcumulada_3, *nuevaAptitud_3, *nuevaDistancias_3;
    int *mejorCamino_3, *pobAuxiliar_3;

    double *distancias_4, *aptitud_4, *puntuacion_4,
           *puntAcumulada_4, *nuevaAptitud_4, *nuevaDistancias_4;
    int *mejorCamino_4, *pobAuxiliar_4;


```

```
//parametros del algotirmo
double probCruce = 0.8;
double probMutacion = 0.1;
double longitud_1,longitud_2,longitud_3,longitud_4,mejor_long;
double longitudes[2];

coordenadasX= ( double * ) malloc ( nCiudades * sizeof ( double ) );
coordenadasY= ( double * ) malloc ( nCiudades * sizeof ( double ) );

//Se guardan las coordenas extraida de los archivos
createCoordinateVectors(coordenadasX, COORDINATES_X);
createCoordinateVectors(coordenadasY, COORDINATES_Y);
```

```
// reserva de memoria para variables din□micas
    padre1 = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    padre2 = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    hijo1 = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    hijo2 = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    distancias = ( double * ) malloc ( tamPoblacion * sizeof ( double ) );
    aptitud = ( double * ) malloc ( tamPoblacion * sizeof ( double ) );
    int tamSubPoblacion = tamPoblacion/4;
```

```
//Punteros de la isla 1
    mejorCamino_1 = ( int * ) malloc ( nCiudades * sizeof ( int ) );
    distancias_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    nuevaDistancias_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    aptitud_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    nuevaAptitud_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    puntuacion_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    puntAcumulada_1 = ( double * ) malloc ( tamSubPoblacion * sizeof ( double ) );
    pobAuxiliar_1 = ( int * ) malloc ( tamSubPoblacion * nCiudades * sizeof ( int ) );
```

```
//punteros de la poblacion y de las subpoblaciones
    poblacion = ( int * ) malloc ( tamPoblacion * nCiudades * sizeof ( int ) );
    isla1 = ( int * ) malloc ( tamSubPoblacion * nCiudades * sizeof ( int ) );
    isla2 = ( int * ) malloc ( tamSubPoblacion * nCiudades * sizeof ( int ) );
    isla3 = ( int * ) malloc ( tamSubPoblacion * nCiudades * sizeof ( int ) );
    isla4 = ( int * ) malloc ( tamSubPoblacion * nCiudades * sizeof ( int ) );

```

- De la población inicial se subdividió en 4 poblaciones

```
// Generacion del problema
matrizDistancias = ( double * ) malloc ( nCiudades * nCiudades * sizeof ( double ) );
crearMatrizDistancia(matrizDistancias, coordenadasX, coordenadasY, nCiudades);

// Generaci□n de poblaci□n inicial
poblacionInicial(poblacion,distancias,aptitud,tamPoblacion,nCiudades,matrizDistancias);
//Se divide la poblacion en 4 subpoblaciones
dividirPoblacion(tamPoblacion,poblacion,isla1,isla2,isla3,isla4,nCiudades);
posMejor_1 = evaluacion(aptitud_1, puntuacion_1, puntAcumulada_1, tamSubPoblacion);
posMejor_2 = evaluacion(aptitud_2, puntuacion_2, puntAcumulada_2, tamSubPoblacion);
posMejor_3 = evaluacion(aptitud_3, puntuacion_3, puntAcumulada_3, tamSubPoblacion);
posMejor_4 = evaluacion(aptitud_4, puntuacion_4, puntAcumulada_4, tamSubPoblacion);

```

- Se itera tantas veces como se haya establecido el número de generaciones, y se evalúa cada 10 iteraciones para hacer el intercambio de los dos mejores individuos de cada isla

```
//Variables para calcular el tiempo de ejecucion
double t1,t2,dif1;
t1 = omp_get_wtime(); // tiempo inicio
for(int generacion = 1; generacion <= numMaxGen; generacion++) {
    if(generacion%10==0) {
        //Intercambio isla 1 con isla 2
        aplicarIntercambio(isla1,isla2,posMejor_1,posMejor_2,nCiudades,tamSubPoblacion);
        posMejor_1 = evaluacion(aptitud_1, puntuacion_1, puntAcumulada_1, tamSubPoblacion);
        posMejor_2 = evaluacion(aptitud_2, puntuacion_2, puntAcumulada_2, tamSubPoblacion);
        //Intercambio isla 2 con isla 3
        aplicarIntercambio(isla2,isla3,posMejor_2,posMejor_3,nCiudades,tamSubPoblacion);
        posMejor_2 = evaluacion(aptitud_2, puntuacion_2, puntAcumulada_2, tamSubPoblacion);
        posMejor_3 = evaluacion(aptitud_3, puntuacion_3, puntAcumulada_3, tamSubPoblacion);
        //Intercambio isla 3 con isla 4
        aplicarIntercambio(isla3,isla4,posMejor_3,posMejor_4,nCiudades,tamSubPoblacion);
        posMejor_3 = evaluacion(aptitud_3, puntuacion_3, puntAcumulada_3, tamSubPoblacion);
        posMejor_4 = evaluacion(aptitud_4, puntuacion_4, puntAcumulada_4, tamSubPoblacion);
        //Intercambio isla 4 con isla 1
        aplicarIntercambio(isla4,isla1,posMejor_4,posMejor_1,nCiudades,tamSubPoblacion);
        posMejor_1 = evaluacion(aptitud_1, puntuacion_1, puntAcumulada_1, tamSubPoblacion);
        posMejor_4 = evaluacion(aptitud_4, puntuacion_4, puntAcumulada_4, tamSubPoblacion);
    }

```

- Se paraleliza una parte de código, de tal forma que cada hilo evolucione una población, para luego intercambiar los mejores, haciendo los procesos de selección, reproducción, mutación y evaluación

```
#pragma omp parallel num_threads(8)
#pragma omp sections
{
    # pragma omp section
    {
        //Genetico en la isla 1
seleccion(isla1, puntAcumulada_1, aptitud_1, distancias_1,// datos entrada
          pobAuxiliar_1,nuevaAptitud_1,nuevaDistancias_1, // Salida de la funcion
          tamSubPoblacion,nCiudades ); // datos entrada
isla1 = pobAuxiliar_1;
distancias_1 = nuevaDistancias_1;
aptitud_1 = nuevaAptitud_1;
reproduccion(isla1,distancias_1,aptitud_1,probCruce,
             matrizDistancias,tamSubPoblacion,nCiudades);
mutados += mutacion(isla1,distancias_1,aptitud_1,probMutacion,
                    matrizDistancias,tamSubPoblacion,nCiudades);

posMejor_1 = evaluacion(aptitud_1,  // datos entrada
                      puntuacion_1,puntAcumulada_1, // Salida de la funcion
                      tamSubPoblacion); // datos entrada

    }
```

Análisis de Resultados obtenidos

$$Calidad = \left(\frac{logitud\ inicial - longitud\ final}{longitud\ inicial}\right) * 100\%$$

Sin modelo de islas

12 Nodos

```
======Fundamentos de computaci□n paralela y distribuida ======
      Algoritmo Gen□tico Simple para el problema del TSP
                Versi□n serial y paralela

N□mero de ciudades: 12
Tama□o de la poblaci□n: 1000
Numero de generaciones: 250

Poblaci□n inicial :
camino :

Longitud del camino : 23993.55

========= INICIO DEL ALGORITMO GENETICO ==========

ALGORITMO GENETICO en serial- tiempo 0.613826

Longitud del camino final : 21430.99
```

Calidad = ((23993-21430.99)/23993)*100%

= 10.6%

15 Nodos

```
======Fundamentos de computaci□n paralela y distribuida ======
      Algoritmo Gen□tico Simple para el problema del TSP
                Versi□n serial y paralela

 N□mero de ciudades: 15
 Tama□o de la poblaci□n: 1000
 Numero de generaciones: 250

Poblaci□n inicial :
camino :

 Longitud del camino : 31791.52

 ========= INICIO DEL ALGORITMO GENETICO ==========

 ALGORITMO GENETICO en serial- tiempo 0.678437

 Longitud del camino final : 22980.97
```

Calidad = ((31791-22980)/31791)*100%

= 27.7%

20 Nodos

```
======Fundamentos de computaci□n paralela y distribuida ======
      Algoritmo Gen□tico Simple para el problema del TSP
                Versi□n serial y paralela

 N□mero de ciudades: 20
 Tama□o de la poblaci□n: 1000
 Numero de generaciones: 250

Poblaci□n inicial :
camino :

 Longitud del camino : 43125.03

 ========= INICIO DEL ALGORITMO GENETICO ==========

 ALGORITMO GENETICO en serial- tiempo 0.872472

 Longitud del camino final : 22375.69
```

Calidad = ((43125-22375)/43125)*100%

= 48.1%

Con modelo de islas

12 Nodos

```
======Fundamentos de computaci□ paralela y distribuida ======
      Algoritmo Gen□tico Simple para el problema del TSP
                Versi□ serial y paralela

N□mero de ciudades: 12
Tama□ de la poblaci□: 1000
Numero de subpoblaciones: 4
Tama□ de las subpoblaci□es: 250
Numero de generaciones: 250
Criterio de intercambio de individuos: modelo de anillo, en donde el intercambio se realizara cada 10 generaciones


Longitud del camino : 243080.33

========= INICIO DEL ALGORITMO GENETICO CON ISLAS==========

ALGORITMO GENETICO en serial- tiempo 0.982117

Longitud del camino final isla 1: 243080.33

Longitud del camino final isla 2: 219576.31

Longitud del camino final isla 3: 252594.89

Longitud del camino final isla 4: 223584.11

Longitud del camino final (mejor camino): 219576.31

Total sujetos mutados : 22697
```

Calidad = ((243080-219576)/243080)*100%

= 9.6%

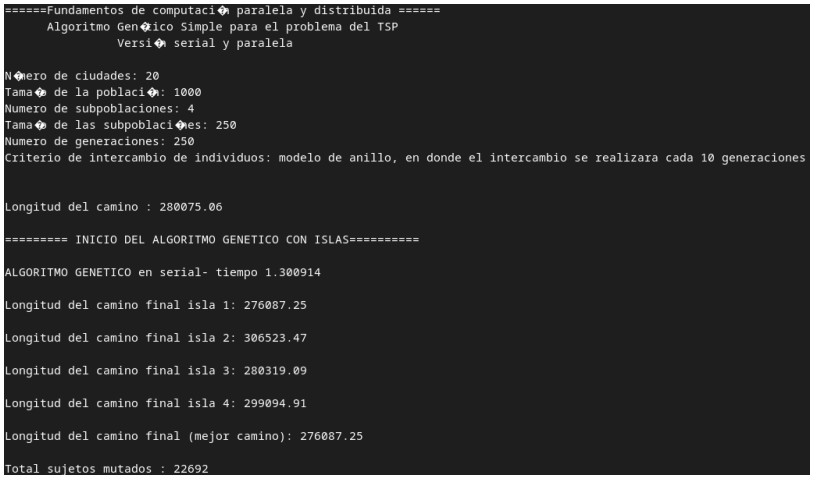
15 Nodos

```
======Fundamentos de computaci□ paralela y distribuida ======
      Algoritmo Gen□tico Simple para el problema del TSP
                Versi□ serial y paralela

N□mero de ciudades: 15
Tama□ de la poblaci□: 1000
Numero de subpoblaciones: 4
Tama□ de las subpoblaci□es: 250
Numero de generaciones: 250
Criterio de intercambio de individuos: modelo de anillo, en donde el intercambio se realizara cada 10 generaciones


Longitud del camino : 280546.06

========= INICIO DEL ALGORITMO GENETICO CON ISLAS==========

ALGORITMO GENETICO en serial- tiempo 1.189569

Longitud del camino final isla 1: 276938.97

Longitud del camino final isla 2: 273433.09

Longitud del camino final isla 3: 260758.66

Longitud del camino final isla 4: 269359.81

Longitud del camino final (mejor camino): 260758.66

Total sujetos mutados : 22332
```

Calidad = ((280546-260758)/280546)*100%

= 7%

20 Nodos



Calidad = ((280075-276087)/280075)*100%

= 1%

Resultados Obtenidos

Se puede concluir que en el algoritmo serial, tuvo una mejor solución que la serializada y mejor tiempo de ejecución, a medida que el número de nodos se iban aumentando se encontraba una mejor solución.