CENTRO DE CIENCIAS BÁSICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

PARALELIZACIÓN DE ALGORITMOS

9° "A"

PROYECTO FINAL

Profesor: Julio César Ponce Gallegos

Alumnos:

Adame Michelle Stephanie

Espinoza Sánchez Joel Alejandro

Gómez Garza Dariana

González Arenas Fernando Francisco

Fecha de Entrega: Aguascalientes, Ags.**, 9** de diciembre de 2022

Entrega del Proyecto (Secuencial)

En este proyecto se aborda el Problema de Ruteo de Vehículos (abreviado como VRP del inglés, Vehicle Routing Problem) el cual intentará resolverse por medio del Algoritmo de Optimización por Colonia de Hormigas (abreviado como ACO del inglés, Ant Colony Optimization).

La prueba de este proyecto se realizó a base de una prueba de rendimiento especializada (conocidas como benchmarks) que están probados en el sitio web de la TSPLIB ubicados en el siguiente sitio:

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

En el sitio anterior se encuentra un aparato dedicado al VRP como se observa a continuación:

Capacitated vehicle routing problem (CVRP)

We are given n-1 nodes, one depot and distances from the nodes to the depot, as well as between nodes. All nodes have demands which can be satisfied by the depot. For delivery to the nodes, trucks with identical capacities are available. The problem is to find tours for the trucks of minimal total length that satisfy the node demands without violating truck capacity constraint. The number of trucks is not specified. Each tour visits a subset of the nodes and starts and terminates at the depot. (Remark: In some data files a collection of alternate depots is given. A CVRP is then given by selecting one of these depots.)

CVRP data

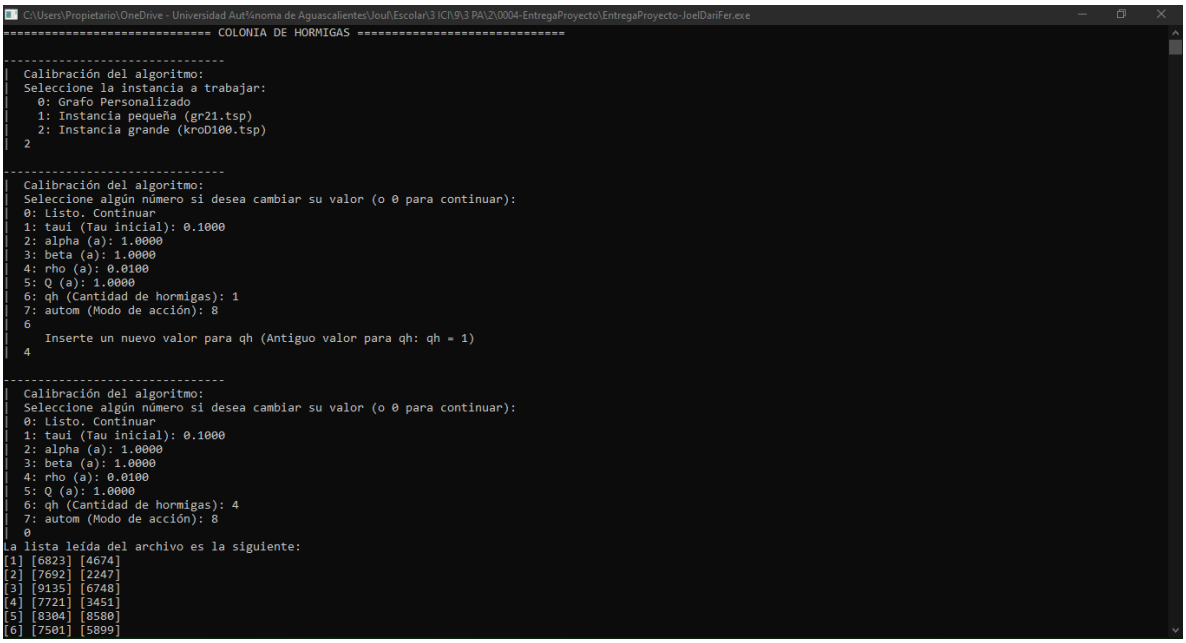
Para la solución de este problema, asistiendo a este link se eligió el problema denominado como "`att48.vrp`" y se preparó un código especializado para esta problemática (véase anexo 1).

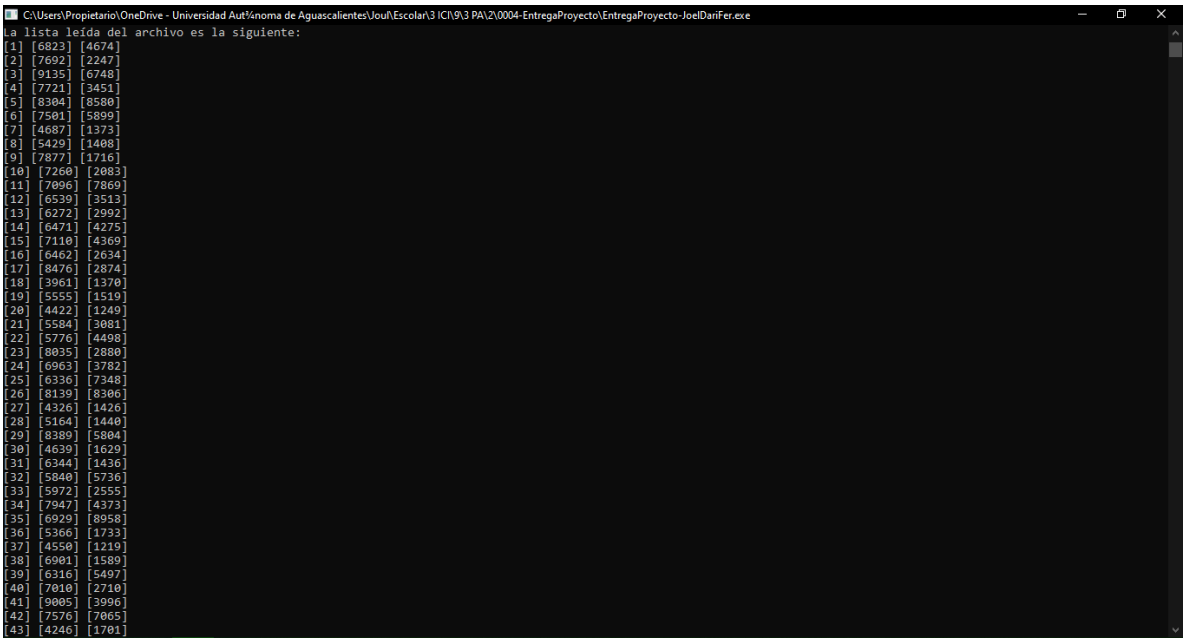
El benchmark iba a dejarse de manera estática, es decir sólo se consideraría este ejemplo con la siguiente información expuesta en el sitio:

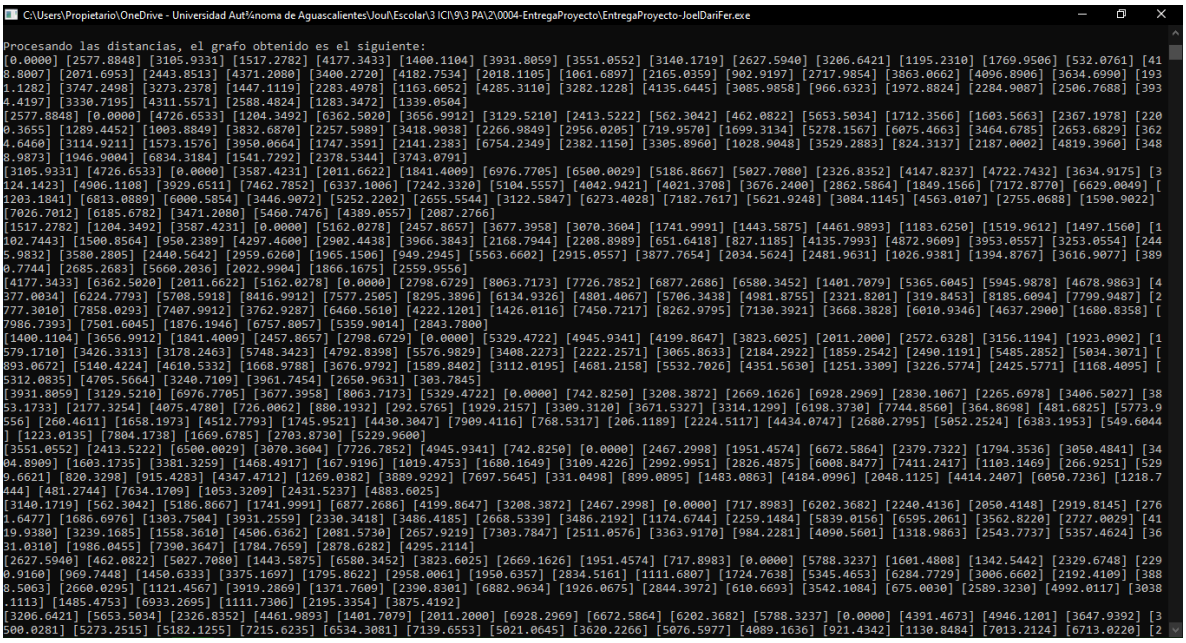
- att48.vrp - Received from Rinaldi,Yarrow/Araque, min. no of trucks: 4, best value: 12649

No se tomarán más benchmarks para centrarse en el diseño del experimento concretamente.

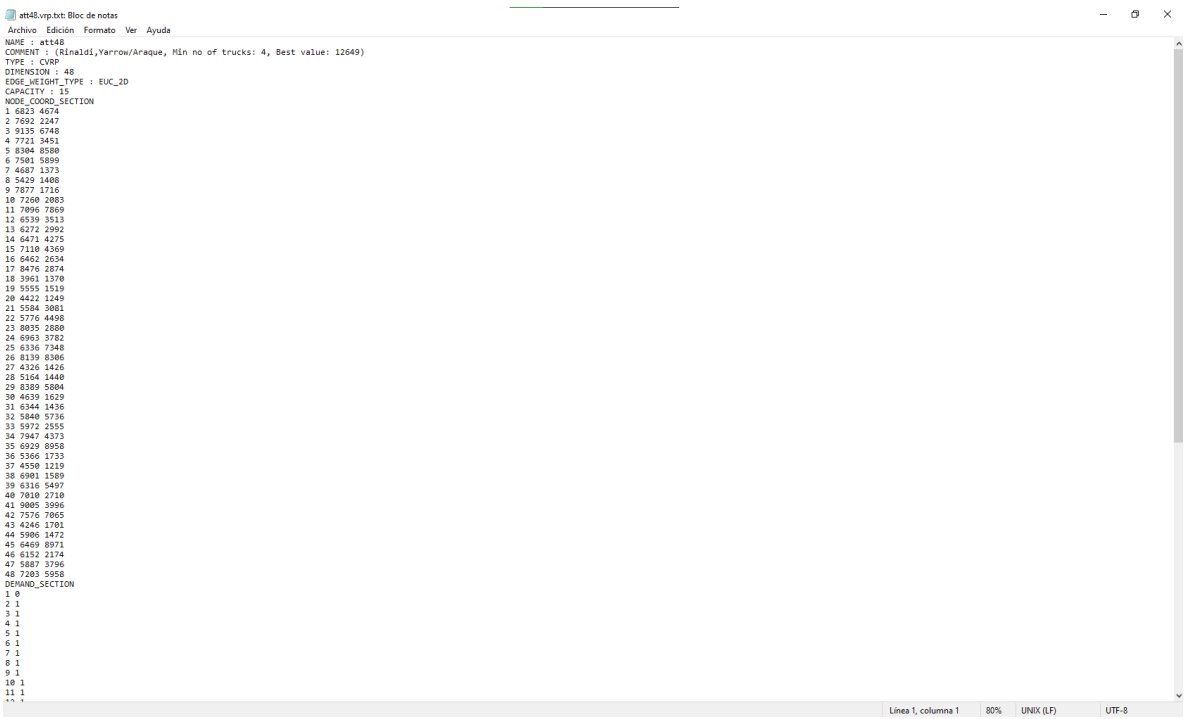
El código elaborado permite personalizar ampliamente el algoritmo así como cargar la instancia de la benchmark. Puede observarse a continuación:


C:\Users\Propietario\OneDrive - Universidad Aut¼noma de Aguascalientes\Joul\Escolar\3 ICI\9\3 PA\2\0004-EntregaProyecto\EntregaProyecto-JoelDariFer.exe
========== COLONIA DE HORMIGAS ==========
Calibración del algoritmo:
Seleccione la instancia a trabajar:
0: Grafo Personalizado
1: Instancia pequeña (gr21.tsp)
2: Instancia grande (kroD100.tsp)
2
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: taui (Tau inicial): 0.1000
2: alpha (a): 1.0000
3: beta (a): 1.0000
4: rho (a): 0.0100
5: Q (a): 1.0000
6: qh (Cantidad de hormigas): 1
7: autom (Modo de acción): 8
6
Inserte un nuevo valor para qh (Antiguo valor para qh: qh = 1)
4
Calibración del algoritmo:
Seleccione algún número si desea cambiar su valor (o 0 para continuar):
0: Listo. Continuar
1: taui (Tau inicial): 0.1000
2: alpha (a): 1.0000
3: beta (a): 1.0000
4: rho (a): 0.0100
5: Q (a): 1.0000
6: qh (Cantidad de hormigas): 4
7: autom (Modo de acción): 8
0
La lista leída del archivo es la siguiente:
[1] [6823] [4674]
[2] [7692] [2247]
[3] [9135] [6748]
[4] [7721] [3451]
[5] [8304] [8580]
[6] [7501] [5899]

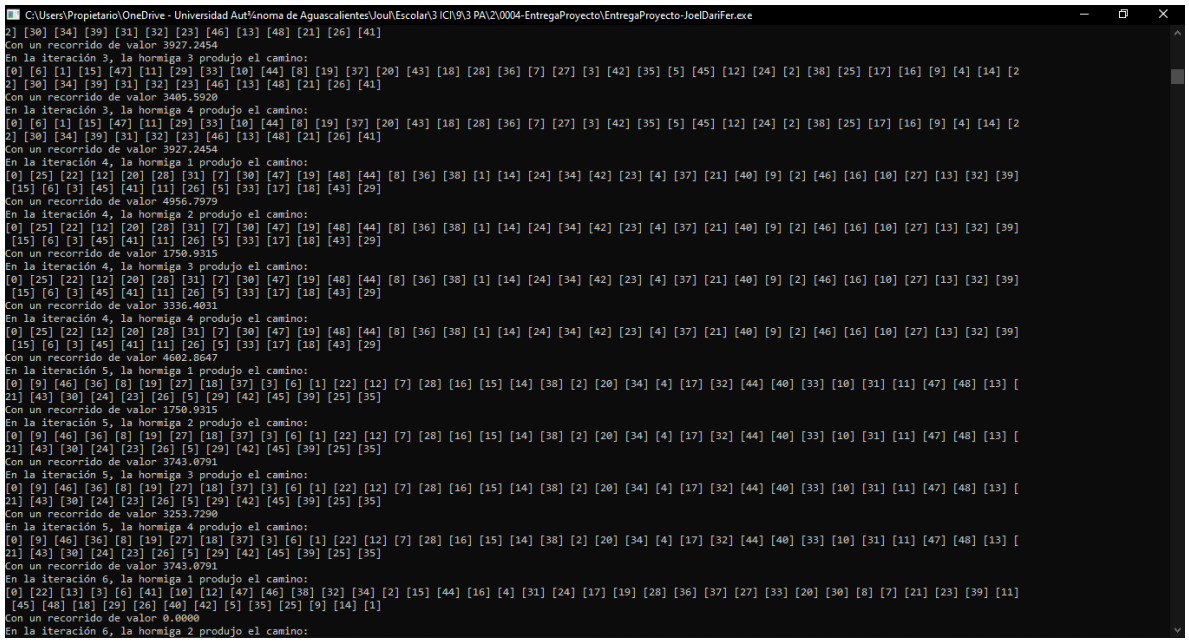

C:\Users\Propietario\OneDrive - Universidad Aut¼noma de Aguascalientes\Joul\Escolar\3 ICI\9\3 PA\2\0004-EntregaProyecto\EntregaProyecto-JoelDariFer.exe
La lista leída del archivo es la siguiente:
[1] [6823] [4674]
[2] [7692] [2247]
[3] [9135] [6748]
[4] [7721] [3451]
[5] [8304] [8580]
[6] [7501] [5899]
[7] [4687] [1373]
[8] [5429] [1408]
[9] [7877] [1716]
[10] [7260] [2083]
[11] [7096] [7869]
[12] [6539] [3513]
[13] [6272] [2992]
[14] [6471] [4275]
[15] [7110] [4369]
[16] [6462] [2634]
[17] [8476] [2874]
[18] [3961] [1370]
[19] [5555] [1519]
[20] [4422] [1249]
[21] [5584] [3081]
[22] [5776] [4498]
[23] [8035] [2880]
[24] [6963] [3782]
[25] [6336] [7348]
[26] [8139] [8306]
[27] [4326] [1426]
[28] [5164] [1440]
[29] [8389] [5804]
[30] [4639] [1629]
[31] [6344] [1436]
[32] [5840] [5736]
[33] [5972] [2555]
[34] [7947] [4373]
[35] [6929] [8958]
[36] [5366] [1733]
[37] [4550] [1219]
[38] [6901] [1589]
[39] [6316] [5497]
[40] [7010] [2710]
[41] [9005] [3996]
[42] [7576] [7065]
[43] [4246] [1701]



Puede observarse el archivo a continuación:



En el ejemplo ejecutado anteriormente se obtuvieron los siguientes resultados:



Es decir no encontró los valores óptimos pero tuvo ciertas aproximaciones importantes. Cabe resaltar que el modelo se invirtió de una minimización a una maximización para comodidad de los estudiantes, quienes hemos elaborado programas de este estilo a diferencia de la minimización.

Descripción del Algoritmo (Secuencial)

Respecto al algoritmo de Colonia de Hormigas la Optimización por medio de Colonia de Hormigas (ACO), es una Metaheurística destinada originalmente a dar solución a problemas de optimización combinatoria.

Esta técnica se basa en el comportamiento estructurado de las colonias de hormigas, donde individuos muy simples se comunican entre sí por medio de una sustancia química denominada feromona, estableciendo el camino óptimo (mínimo) entre el hormiguero y su fuente de alimento.

En términos generales el algoritmo de colonia de hormigas consiste en la creación de distintas generaciones de hormigas que recorrerán el grafo del problema a resolver, construyendo en cada caso una solución por hormiga. Los movimientos que sigue una hormiga en un punto de tiempo específico dependen de información local (conocida como visibilidad) y de información global (feromona depositada por cada hormiga que ha transitado por una arista del grafo).

La siguiente figura muestra el algoritmo general de ACO (Ant Colony Optimization):

Algoritmo 1 Metaheurística ACO

```
Establecer parámetros, inicializar rastros de feromona
while (no se cumpla condición de terminación) do
  ConstruirSolucionesporHormigas
  AplicarBúsquedaLocal { Opcional }
  ActualizarFeromona
end while
```

De acuerdo con este algoritmo, la metaheurística consiste en una fase de inicialización de parámetros, seguida de una iteración sobre tres componentes: la construcción de soluciones por cada una de las hormigas, la mejora de la solución mediante algún mecanismo de búsqueda local (opcional) y la actualización de la feromona.

La construcción de una solución por cada hormiga se realiza a partir de un conjunto finito de componentes de una solución disponible. Por ejemplo, al recorrer los nodos de un grafo, cada hormiga inicia en un vértice (origen) y en cada paso se va agregando un nuevo nodo a la solución hasta llegar a la meta (destino).

La k – ésima hormiga se moverá del vértice i al j con una probabilidad dada por:

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}$$

Donde τ_{ij} es la es la cantidad de feromonas depositadas en la transición del estado i al j, α es un parámetro para controlar la influencia de τ_{ij}; η_{ij} es la conveniencia del estado de transición $i-j$ (conocimiento a priori, típicamente $\frac{1}{d_{ij}}$), donde d es la distancia o costo de la transición y β es un parámetro para controlar la influencia de η_{ij}.

Cuando las k hormigas han completado una solución, se debe actualizar los rastros de feromona por:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{k} \Delta\tau_{ij}^{k}$$

Donde:

τ_{ij} es la feromona depositada en la arista $i-j$, ρ es el coeficiente de evaporación de la feromona y $\Delta\tau_{ij}^{k}$ es la cantidad de feromona depositada por la k – ésima hormiga, típicamente dada por:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} \; si\ k\ usó\ la\ arista\ i-j \\ 0\ si\ k\ no\ usó\ la\ arista\ i-j \end{cases}$$

Donde L_k es el costo de la solución de la k – ésima hormiga y Q es una constante.

Diseño del Experimento (Secuencial)

El diseño del experimento partiría desde los siguientes valores estándares:

- $\alpha = 1.$
- $\beta = 1.$
- $\tau = 0.1.$
- $\rho = 0.01.$

La idea del diseño del experimento sería variar los valores y reportar los resultados en cada iteración realizada. Los escenarios planteados serían en un área de variación de cuatro dimensiones donde los parámetros varían de la siguiente manera:

- $\alpha \in \{1, 2, 3, 4\}.$
- $\beta \in \{1, 2, 3, 4\}.$
- $\tau \in \{0.1, 0.2, 0.3, 0.4\}.$
- $\rho \in \{0.01, 0.02, 0.03, 0.04\}.$

Así se observaría cuál de estas variaciones trae mejores resultados en comparación a los reportados en el sitio de la TSPLIB.

Resultados del Diseño del Experimento (Secuencial)

El diseño del experimento parte de una experimentación con las siguientes variaciones:

- $\alpha \in \{1, 2, 3\}$.
- $\beta \in \{1, 2, 3\}$.
- $\tau \in \{0.1, 0.2, 0.3\}$.
- $\rho \in \{0.01, 0.02, 0.03\}$.

Cada caso es reportado a continuación mostrando la diferencia existente entre los resultados del algoritmo propio y el demostrado en el sitio de la TSPLIB:

Valores	Prueba 1	Prueba 2	Prueba 3	Prueba 4	Promedio
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.01.$	4390.183184	4392.903616	4398.8804	4408.113536	4397.520184
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.01.$	4383.410881	4386.904576	4393.962721	4404.585316	4392.215874
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.01.$	4382.835396	4388.996025	4401.149376	4419.295449	4398.069062
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.01.$	4391.822649	4397.446521	4408.397241	4424.674809	4405.585305
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.01.$	4386.708964	4391.456896	4400.6689	4414.344976	4398.294934
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.01.$	4383.239121	4386.474496	4393.491121	4404.288996	4391.873434
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.01.$	4393.172889	4400.031556	4413.022225	4432.144896	4409.592892
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.01.$	4391.859329	4397.230016	4407.524225	4422.741956	4404.838882

$\alpha = 3.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.01.$	4387.1236	4394.088336	4407.418384	4427.113744	4403.936016
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.01.$	4393.815409	4400.273809	4413.169081	4432.501225	4409.939881
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.01.$	4387.127449	4389.452356	4394.700625	4402.872256	4393.538172
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.01.$	4378.394384	4382.515625	4391.118884	4404.204161	4389.058264
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.01.$	4394.643204	4399.461569	4408.992384	4423.235649	4406.583202
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.01.$	4384.7225	4389.895184	4400.048036	4415.181056	4397.461694
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.01.$	4382.385641	4385.452164	4391.579025	4400.766224	4390.045764
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.01.$	4395.2304	4398.080025	4404.1809	4413.533025	4402.756088
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.01.$	4388.813604	4395.279204	4408.196036	4427.5641	4404.963236
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.01.$	4381.599076	4385.695889	4393.6736	4405.532209	4391.625194
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.01.$	4396.350464	4400.343056	4408.787776	4421.684624	4406.79148
$\alpha = 2.$	4392.558009	4397.317636	4406.938225	4421.419776	4404.558412

$\beta = 1.$ $\tau = 0.3.$ $\rho = 0.01.$					
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.01.$	4385.275625	4388.709476	4396.068929	4407.353984	4394.352004
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.01.$	4397.273529	4400.728761	4408.148921	4419.534009	4406.421305
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.01.$	4391.5184	4395.313929	4402.792356	4413.953681	4400.894592
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.01.$	4384.228484	4387.090564	4393.229444	4402.645124	4391.798404
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.01.$	4398.295936	4401.489424	4408.188864	4418.394256	4406.59212
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.01.$	4393.085764	4399.823209	4412.720704	4431.778249	4409.351982
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.01.$	4385.368449	4388.6864	4395.452289	4405.666116	4393.793314
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.02.$	4390.541696	4395.216656	4404.684224	4418.9444	4402.346744
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.02.$	4388.094116	4394.656289	4407.142144	4425.551681	4403.861058
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.02.$	4383.343396	4386.526884	4393.0489	4402.909444	4391.457156
$\alpha = 1.$ $\beta = 2.$	4391.465124	4395.343056	4403.152196	4414.892544	4401.21323

$\tau = 0.1.$ $\rho = 0.02.$					
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.02.$	4382.141376	4384.696164	4390.456464	4399.422276	4389.17907
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.02.$	4379.1236	4385.628644	4397.927296	4416.019556	4394.674774
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.02.$	4392.815409	4398.507601	4409.631601	4426.187409	4406.785505
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.02.$	4390.301401	4393.111696	4398.874441	4407.589636	4397.469294
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.02.$	4380.185761	4382.812329	4388.543929	4397.380561	4387.230645
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.02.$	4393.49	4398.202961	4407.915044	4422.626249	4405.558564
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.02.$	4389.329476	4393.149369	4401.25	4413.631369	4399.340054
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.02.$	4381.180625	4383.996361	4390.223369	4399.861649	4388.815501
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.02.$	4394.361201	4397.775249	4404.791225	4415.409129	4403.084201
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.02.$	4385.030225	4390.996025	4402.275625	4418.869025	4399.292725
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.2.$	4382.238769	4389.300161	4402.613201	4422.177889	4399.082505

$\rho = 0.02.$					
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.02.$	4395.355216	4398.530641	4404.998244	4414.758025	4403.410532
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.02.$	4392.622521	4397.517801	4407.280281	4421.909961	4404.832641
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.02.$	4390.242064	4392.999824	4398.832784	4407.740944	4397.453904
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.02.$	4396.833569	4402.185169	4412.491721	4427.753225	4409.815921
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.02.$	4389.515524	4393.318084	4400.819844	4412.020804	4398.918564
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.02.$	4383.405769	4387.477456	4395.924025	4408.745476	4393.888182
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.02.$	4397.373321	4400.396649	4406.455625	4415.550249	4404.943961
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.02.$	4388.9409	4394.974881	4406.593344	4423.796289	4403.576354
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.02.$	4385.253009	4388.556996	4395.686361	4406.641104	4394.034368
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.02.$	4398.301401	4401.352561	4407.690769	4417.316025	4406.165189
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.02.$	4390.535824	4394.990756	4403.957696	4417.436644	4401.73023

$\alpha = 3.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.02.$	4384.238144	4387.305124	4393.909904	4404.052484	4392.376414
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.03.$	4391.087849	4397.463824	4409.545241	4427.3321	4406.357254
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.03.$	4383.356409	4386.415104	4392.603801	4401.9225	4391.074454
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.1.$ $\rho = 0.03.$	4375.855625	4381.76	4393.275625	4410.4025	4390.323438
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.03.$	4391.753424	4397.579225	4409.164644	4426.509681	4406.251744
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.03.$	4384.641601	4390.180196	4401.397241	4418.292736	4398.627944
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.1.$ $\rho = 0.03.$	4383.367236	4386.485689	4392.784384	4402.263321	4391.225158
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.03.$	4392.253009	4395.157729	4401.308601	4410.705625	4399.856241
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.03.$	4386	4392.273809	4404.307236	4422.100281	4401.170332
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.1.$ $\rho = 0.03.$	4377.0961	4379.307361	4384.441984	4392.499969	4383.336354
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.03.$	4393.850084	4399.046681	4408.968016	4423.614089	4406.369718
$\alpha = 2.$	4383.6561	4388.230369	4397.167696	4410.468081	4394.880562

$\beta = 1.$ $\tau = 0.2.$ $\rho = 0.03.$					
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.2.$ $\rho = 0.03.$	4379.414736	4382.644281	4389.074276	4398.704721	4387.459504
$\alpha = 1.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.03.$	4394.561001	4399.308416	4408.891881	4423.311396	4406.518174
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.03.$	4384.4356	4387.984016	4395.102224	4405.790224	4393.328016
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.2.$ $\rho = 0.03.$	4379.603729	4384.143824	4393.130081	4406.5625	4390.860034
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.03.$	4395.64	4400.313025	4409.5161	4423.249225	4407.179588
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.03.$	4393.753424	4399.3504	4410.405584	4426.918976	4407.607096
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.2.$ $\rho = 0.03.$	4390.442225	4394.528384	4402.895281	4415.542916	4400.852202
$\alpha = 1.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.03.$	4397.052676	4403.817616	4416.848356	4436.144896	4413.465886
$\alpha = 2.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.03.$	4390.537289	4395.058001	4404.175225	4417.888961	4401.914869
$\alpha = 3.$ $\beta = 1.$ $\tau = 0.3.$ $\rho = 0.03.$	4387.9216	4394.257636	4406.607184	4424.970244	4403.439166
$\alpha = 1.$ $\beta = 2.$	4397.737881	4403.446521	4414.799961	4431.798201	4411.945641

$\tau = 0.3.$ $\rho = 0.03.$					
$\alpha = 2.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.03.$	4396.287296	4399.61	4406.653696	4417.418384	4404.992344
$\alpha = 3.$ $\beta = 2.$ $\tau = 0.3.$ $\rho = 0.03.$	4390.950625	4397.193124	4409.263321	4427.161216	4406.142072
$\alpha = 1.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.03.$	4398.310249	4401.568321	4408.374841	4418.729809	4406.745805
$\alpha = 2.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.03.$	4389.132096	4396.145316	4409.566736	4429.396356	4406.060126
$\alpha = 3.$ $\beta = 3.$ $\tau = 0.3.$ $\rho = 0.03.$	4386.381924	4389.928324	4397.195716	4408.1841	4395.422516

Puede mejorarse este diseño de experimento con mejores restricciones para poder interpretar de mejor manera qué configuración de variables es la óptima.

Diseño del Algoritmo (Paralelo)

Para el diseño de este algoritmo se tuvo que instala el CUDA Toolkit.

GeForce and TITAN Products

GPU	Compute Capability
GeForce RTX 3090 Ti	8.6
GeForce RTX 3090	8.6
GeForce RTX 3080 Ti	8.6
GeForce RTX 3080	8.6
GeForce RTX 3070 Ti	8.6
GeForce RTX 3070	8.6
Geforce RTX 3060 Ti	8.6
Geforce RTX 3060	8.6
GeForce GTX 1650 Ti	7.5
NVIDIA TITAN RTX	7.5
Geforce RTX 2080 Ti	7.5
Geforce RTX 2080	7.5
Geforce RTX 2070	7.5
Geforce RTX 2060	7.5
NVIDIA TITAN V	7.0

GeForce Notebook Products

GPU	Compute Capability
GeForce RTX 3080 Ti	8.6
GeForce RTX 3080	8.6
GeForce RTX 3070 Ti	8.6
GeForce RTX 3070	8.6
GeForce RTX 3060	8.6
GeForce RTX 3050 Ti	8.6
GeForce RTX 3050	8.6
Geforce RTX 2080	7.5
Geforce RTX 2070	7.5
Geforce RTX 2060	7.5
GeForce GTX 1080	6.1
GeForce GTX 1070	6.1
GeForce GTX 1060	6.1
GeForce GTX 980	5.2
GeForce GTX 980M	5.2

Visitando la página de NVIDIA se encontró que el equipo con el que se haría este trabajo posee una tarjeta gráfica GeForce RTX 3050, implicando con esto una capacidad de cómputo de la versión 8.6:

GeForce RTX 3050 Ti	8.6
GeForce RTX 3050	8.6

La versión más reciente (CUDA Toolkit 11) soporta esta capacidad de cómputo, por lo que se procedió.

Primeramente se tomó el código secuencial tal cual y se le debieron de hacer ciertas adecuaciones puesto que en el IDE Visual Studio no se podían ejecutar las funciones como se esperaba, pero posterior a esto, los cambios fueron meramente adaptativos a la estructura de CUDA. A continuación se presentan algunos ejemplos de cambios realizados, donde a la izquierda se observa el código en secuencial y a la derecha el código en paralelo:

```
//Cargamos las librerias
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>
#include <string.h>


//void setInstance21(int qv, float graph[qv][qv]);
```

```
//Cargamos las librerias
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

cudaError_t nodeDistanceCuda(int *graph, int *list, int num);
cudaError_t visionInitCuda(float *vision, float *pheromone, int *graph, float alp

//void setInstance21(int qv, float graph[qv][qv]);

__global__ void addKernelnodeDistanceCuda(int *graph, int *list)
{
    int i = threadIdx.x;
    int j = threadIdx.x;
    graph[i][j] = sqrt((pow((list[i][1] - list[j][1]),2))+(pow((list[i][2] - list
}

__global__ void addKernelvisionInitCuda(float *vision, float *pheromone, int *gra
{
    int i = threadIdx.x;
    int j = threadIdx.x;
    if(graph[i][j] == 0)
    {
        vision[i][j] = 0;
    }
    else
    {
        vision[i][j] = (pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
    }
}
```

```
        //Imprimimos lo que obtuvimos de la lectura de archivos
        printf("La lista leída del archivo es la siguiente:\n");
        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 3; j++)
            {
                printf("[%d] ",list[i][j]);
            }
            printf("\n");
        }
        printf("\n");

        //Calculamos las distancias entre los nodos y éstas serán guardadas en


        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 48; j++)
            {
                graph[i][j] = sqrt((pow((list[i][1] - list[j][1]),2))+(pow((list
            }
        }


        //Imprimimos el grafo con el que trabajaremos
```

```
        //Imprimimos lo que obtuvimos de la lectura de archivos
        printf("La lista leída del archivo es la siguiente:\n");
        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 3; j++)
            {
                printf("[%d] ",list[i][j]);
            }
            printf("\n");
        }
        printf("\n");

        //Calculamos las distancias entre los nodos y éstas serán guardadas e
        cudaError_t cudaStatus = nodeDistanceCuda(graph, list, 48);
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "nodeDistanceCuda failed!");
            return 1;
        }

        /*
        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 48; j++)
            {
                graph[i][j] = sqrt((pow((list[i][1] - list[j][1]),2))+(pow((l
            }
        }
        */

        //Imprimimos el grafo con el que trabajaremos
```

```
cudaError_t nodeDistanceCuda(int *graph, int *list, int num)
{
    int *dev_graph = 0;
    int *dev_list = 0;
    cudaError_t cudaStatus;

    // Se elige la GPU de ejecución
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU in
        goto Error;
    }

    // Se fijan buffers de GPU para dos arreglos (un input, un output)
    cudaStatus = cudaMalloc((void**)&dev_graph, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_list, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    // Se copia el arreglo de entrada de memoria host a buffer de GPU
    cudaStatus = cudaMemcpy(dev_list, list, size * sizeof(int), cudaMemcpyHostToD
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
```

La captura y comparación de tiempos no fue posible realizarla.

Conclusiones

Michelle Stephanie Adame: En el transcurso de este proyecto aclaramos conocimientos anteriores, haciendo un poco más de investigación teórica para definir algunos aspectos del algoritmo importantes. Asimismo observamos cómo aumenta la dificultad cuando el número de hormigas va aumentando en una colonia. Personalmente es un algoritmo sencillo para su comprensión. Sin embargo, es algo más tardado de elaborar manualmente, y esto es dependiendo el número de hormigas que se establecerán en la colonia, ya que se debe hacer un recorrido diferente por cada hormiga y por lo tanto se debe realizar el mismo proceso una por una. Pese a eso, queda claro cómo se inspiraron al crear este algoritmo, ya que nos deja como ejemplo como es que las hormigas naturales, de la vida real, trabajan para llegar por ejemplo, a su comida que tienen almacenada.

Joel Alejandro Espinoza Sánchez: La colonia de hormigas es un mecanismo heurístico muy útil cuando la exploración simultánea puede ser un gran factor. La implementación de este algoritmo nos permitió profundizar en la colonia de dos o más hormigas, así como la modificación de la trayectoria del grafo, el cual debía recorrer completamente el grafo. Estas consideraciones nos permitieron ampliar las aplicaciones de la colonia de hormigas.

Asimismo, pudimos observar cómo se comportaba el algoritmo mediante la prueba de distintas configuraciones de variables, pues con menor factor de evaporación, las feromonas se mantenían más, sin embargo, de ser un factor notorio, las feromonas tendían a cero rápidamente. Así también los factores de influencia de las feromonas y la visualización jugaban un papel tan importante que al final se decidió mantener en una igualdad en uno, pues de esta manera afecarían de forma equivalente al procedimiento.

Dariana Gómez Garza: A lo largo de este proyecto pudimos reforzar los conocimientos previos, ya que se tuvo que hacer un poco más de investigación para definir algunos aspectos del algoritmo. También nos dimos cuenta de cómo va creciendo la dificultad conforme el número de hormigas va aumentando en una colonia.

A mi parecer es un algoritmo muy sencillo de entender, lo único es que si es algo más tardado de hacer (manualmente) dependiendo la cantidad de hormigas que estarán en la colonia, ya que se debe hacer un recorrido diferente por cada hormiga y por lo tanto se debe realizar el mismo proceso una por una. A pesar de eso queda muy claro como se inspiraron al crear este algoritmo, ya que nos deja como ejemplo como es que las hormigas naturales, de la vida real, trabajan para llegar por ejemplo, a su comida que tienen almacenada.

Fernando Francisco González Arenas: La colonia de hormigas es un algoritmo de optimización muy útil, el cual emula la organización de las hormigas biológicas para encontrar comida utilizando la ruta más corta. Esto lo hacen por medio de las feromonas que van dejando en el camino y que las otras hormigas de la colonia van siguiendo para encontrar el camino mas corto.

El algoritmo es muy interesante, ya que simula con fórmulas y variables la forma de actuar de las hormigas, como van dejando feromonas en el camino que les parece el mejor y así hacen que las otras hormigas (que en este caso son iteraciones del algoritmo), las sigan y se encuentre el objetivo deseado. Yo opino que este algoritmo nos podra ser útil en el futuro para problemas de optimización, ya sea en nuestra vida académica o laboral.

Bibliografía

- Alonso, S. et ál. (2004), La metaheurística de optimización basada en colonias de hormigas: modelos y nuevos enfoques, Granada, Departamento de Ciencias de la Computación e Inteligencia Artificial, E.T.S. Universidad de Granada.

- Ponce, J., Padilla, F. y Ochoa, C. (2006), Algoritmo de optimización con colonia de hormigas para el problema de la mochila, México, Universidad Autónoma de Aguascalientes.

- Dorigo, M. y Blum, C. (2005), “Ant Colony Optimization Theory: A Survey”,en Lecture Notes in Computer Science, vol. 344, pp. 243-278.

- Ramírez, J. A., Zacarías, H. (2007). Gestión del conocimiento e Innovación en la toma de decisiones en el abastecimiento de librerías. Tesis de Doctorado no publicada. Centro de Estudios Avanzados del IPN.México.

- Bello Pérez, R. (2 de julio de 2008). Teoría de conjuntos aproximados y colonia de hormigas en el contexto de la inteligencia artificial. (R. Millet Luaces, Entrevistador).

- Reinelt, G. (1994), The traveling salesman: computational solutions for TSP applications, Berlin, Springer-Verlag.

- Torres, A (2020) Optimización mediante colonia de hormigas. México: Universidad Autónoma de Aguascalientes.

Anexos

Anexo 1: Contenido del documento `att48.vrp`.

```
NAME : att48
COMMENT : (Rinaldi,Yarrow/Araque, Min no of trucks: 4, Best value: 12649)
TYPE : CVRP
DIMENSION : 48
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 15
NODE_COORD_SECTION
1 6823 4674
2 7692 2247
3 9135 6748
4 7721 3451
5 8304 8580
6 7501 5899
7 4687 1373
8 5429 1408
9 7877 1716
10 7260 2083
11 7096 7869
12 6539 3513
13 6272 2992
14 6471 4275
15 7110 4369
16 6462 2634
17 8476 2874
18 3961 1370
19 5555 1519
20 4422 1249
21 5584 3081
22 5776 4498
23 8035 2880
24 6963 3782
25 6336 7348
26 8139 8306
27 4326 1426
28 5164 1440
29 8389 5804
30 4639 1629
31 6344 1436
32 5840 5736
33 5972 2555
34 7947 4373
35 6929 8958
36 5366 1733
37 4550 1219
38 6901 1589
```

39 6316 5497
40 7010 2710
41 9005 3996
42 7576 7065
43 4246 1701
44 5906 1472
45 6469 8971
46 6152 2174
47 5887 3796
48 7203 5958
DEMAND_SECTION
1 0
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
21 1
22 1
23 1
24 1
25 1
26 1
27 1
28 1
29 1
30 1
31 1
32 1
33 1
34 1
35 1
36 1
37 1
38 1
39 1

```
40 1
41 1
42 1
43 1
44 1
45 1
46 1
47 1
48 1
DEPOT_SECTION
 1
 -1
EOF
```

Anexo 2: Código de solución de la Colonia de Hormigas para el Problema de Ruteo de Vehículos en Secuencial.

```
/*
    Universidad Autónoma de Aguascalientes

          Centro de Ciencias Básicas
  Departamento de Ciencias de la Computación
         Paralelización de Algoritmos

                    9° "A"

            Tercera Evaluación Parcial
               Entrega del Proyecto

        Dr. Julio César Ponce Gallegos

                  Alumnos:
        Adame Michelle Stephanie
        Espinoza Sánchez Joel Alejandro
        Gómez Garza Dariana
        González Arenas Fernando Francisco

  Fecha de Entrega: 8 de diciembre del 2022
*/

//Cargamos las librerías
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>
#include <string.h>

void setValues1(float alpha, int autom, float beta, int instance, float Q,
int qh, int qv, float rho, float taui,float *alphaP, int *automP, float
*betaP, int *instanceP, float *QP, int *qhP, int *qvP, float *rhoP, float
*tauiP);
void fillMatrix1D(int a, float matrix[a], float n);
void fillMatrix2D(int a, int b, float matrix[a][b], float n);
void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n);
void setInstance21(int qv, float graph[qv][qv]);
void setInstance100(int qv, float graph[qv][qv]);
void   setValues2(int   qv,   int   qh,   float   graph[qv][qv],   float
pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2]);
void  startProcess(int  i,  int  qh,  int  qv,  float  tabu[qh][qv],  float
beginEnd[qh][2], float availableV[qv]);
int acceptVertex(int i, int qh, int qv, int value, float tabu[qh][qv]);
```

```
float getL(int i, int qh, int qv, float graph[qv][qv], float tabu[qh][qv],
float beginEnd[qh][2]);

main()
{
	setlocale(LC_ALL,"");
	srand(time(NULL));

	//1. Declaramos las variables que usaremos
	/*
		qv: Cantidad de vértices del grafo
		qh: Cantidad de hormigas
		tau: Valor de la feromona inicial
		Q: Parámetro
		alpha: Parámetro
		beta: Parámetro
		instance: Instancia que se usará en el recocido, donde:
				Si instance = 0 ejecutará un proceso
personalizado
				Si instance = 1 ejecutará el proceso de la
instancia pequeña
				Si instance = 2 ejecutará el proceso de la
instancia grande
		rho: Parámetro
		autom: Modo de acción (Para autom >=1 se tomará el número de
autom como las repeticiones)
		repeat: Sirve para repetir todo el código nuevamente
		repeat1: Sirve para repetir la colonia en modo manual
		r: Número aleatorio
		accepted: Bandera de aceptación de vértice
		checkIfStuck: Iterador que no tolerará un número determinado
de iteraciones. Si se alcanzan, se interpretará que la hormiga se encerró
		L: Longitud del recorrido completo realizado por la hormiga
	*/
	int qv, qh, autom, instance, repeat, repeat1, repeat2, accepted,
checkIfStuck, h, i, j, k, ip, jp, kp;
	float tau, Q, alpha, beta, rho, L, r;

	printf("============================== COLONIA DE HORMIGAS
==============================\n");

	do
	{
		alpha = 1;
		autom = 8;
		beta = 1;
		Q = 1;
		qh = 1;
		qv = 7;
		repeat = 0;
```

```
            rho = 0.01;
            tau = 0.1;

            //3. Calibramos este primer conjunto de valores del algoritmo
            printf("\n");
            printf("--------------------------------\n");
            printf("|  Calibración del algoritmo:\n");
            printf("|  Seleccione la instancia a trabajar:\n");
            printf("|    0: Grafo Personalizado\n");
            printf("|    1: Instancia pequeña (gr21.tsp)\n");
            printf("|    2: Instancia grande (kroD100.tsp)\n");
            printf("|  ");
            scanf("%d",&instance);

            //2. Calibramos los primeros valores del programa

      setValues1(alpha,autom,beta,instance,Q,qh,qv,rho,tau,&alpha,&autom,
&beta,&instance,&Q,&qh,&qv,&rho,&tau);

            //3. Declaramos más variables
            /*
                  graph: La matriz de adyacencia del grafo que analizaremos

            INT
                  pheromone: La matriz de feromonas

                        FLOAT
                  vision: La matriz de visibilidad

                        FLOAT
                  tabu: La lista tabú

                                    INT
                  beginEnd: Matriz con los valores de inicio y fin de cada
hormiga
      INT
                  availableV: Vector donde cada elemento representa cada
vértice del grafo (1 está disponible o 0 no lo está)      INT
                  prob: Probabilidad individual y acumulada de tomar cada
nodo
      FLOAT
                  dtau: La diferencia de tau que hay por hormiga en cada
nodo
            FLOAT
            */
            float   graph[qv][qv],   pheromone[qv][qv],   vision[qv][qv],
tabu[qh][qv],      beginEnd[qh][2],      availableV[qv],      prob[qv][2],
dtau[qh][qv][qv];

            //5. Calibramos este segundo conjunto de valores del algoritmo
```

```
		if(instance == 1)
		{
			setInstance21(qv,graph);
		}
		if(instance == 2)
		{
			setInstance100(qv,graph);
		}

		//4. Inicializamos las estructuras
		if(instance == 0)
		{
			//Limpiamos graph
			fillMatrix2D(qv,qv,graph,0);
		}

		//Inicializamos pheromone
		fillMatrix2D(qv,qv,pheromone,tau);

		//Limpiamos vision
		fillMatrix2D(qv,qv,vision,0);

		//Limpiamos tabu
		fillMatrix2D(qh,qv,tabu,0);

		//Limpiamos beginEnd
		fillMatrix2D(qh,2,beginEnd,0);

		//Inicializamos availableV
		fillMatrix1D(qv,availableV,1);

		//Limpiamos prob
		fillMatrix2D(qv,2,prob,0);

		//5. Calibramos los demás valores del programa
		setValues2(qv,qh,graph,pheromone,vision,beginEnd);

		//Inicializamos vision
		for(i = 0; i < qv; i++)
		{
			for(j = 0; j < qv; j++)
			{
				if(graph[i][j] == 0)
				{
					vision[i][j] = 0;
				}
				else
				{
					vision[i][j]                                    =
(pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
```

```
                        }
                  }
            }

            printf("\n\n\n\n");

            if(autom == 0)
            {
                  printf("El grafo a evaluar es:\n");
                  for(ip = 0; ip < qv; ip++)
                  {
                        for(jp = 0; jp < qv; jp++)
                        {
                              printf("[%d] ", (int)graph[ip][jp]);
                        }
                        printf("\n");
                  }
                  printf("\n");

                  printf("La matriz de feromonas antes del comienzo del
algoritmo es:\n");
                  for(ip = 0; ip < qv; ip++)
                  {
                        for(jp = 0; jp < qv; jp++)
                        {
                              printf("[%.4f] ", pheromone[ip][jp]);
                        }
                        printf("\n");
                  }
                  printf("\n");

                  printf("La matriz de visibilidad, entonces, es:\n");
                  for(ip = 0; ip < qv; ip++)
                  {
                        for(jp = 0; jp < qv; jp++)
                        {
                              printf("[%.4f] ", vision[ip][jp]);
                        }
                        printf("\n");
                  }
                  printf("\n");
                  getchar();
                  getchar();
            }

            //6. Comenzamos la colonia de hormigas, la cual se repetirá
tantas veces como se haya calibrado autom (si es 0, se decide el paro)
            h = 0;
            repeat2 = 1;
            do
```

```
            {
                  //Inicializamos dtau
                  fillMatrix3D(qh,qv,qv,dtau,0);

                  if(autom == 0)
                  {
                        printf("\n");
                        printf("\n");
                        printf("Comenzamos la %d° iteración\n\n", h + 1);
                  }

                  //El procedimiento de la colonia se hará para cada
hormiga
                  i = 0;
                  do
                  {
                        if(autom == 0)
                        {
                              printf("    Comenzamos a trabajar con la
hormiga %d\n", i + 1);
                              getchar();
                        }

                        //Inicializamos tabu y availableV con los valores
proporcionados por beginEnd
                        startProcess(i,qh,qv,tabu,beginEnd,availableV);

                        if(autom == 0)
                        {
                              printf("    Inicializamos la lista tabú con
el valor del nodo de inicio:\n    ");
                              for(ip = 0; ip < qh; ip++)
                              {
                                    for(jp = 0; jp < qv; jp++)
                                    {
                                          printf("[%d]                  ",
(int)tabu[ip][jp]);
                                    }
                                    printf("\n    ");
                              }

                              printf("\n");
                              printf("    Inicializamos la lista de nodos
disponibles:\n    ");
                              for(ip = 0; ip < qv; ip++)
                              {
                                    printf("[%d] ", (int)availableV[ip]);
                              }
                              printf("\n");
```

```
                        getchar();
                  }

                  //Todavía, el procedimiento de selección de camino
de cada hormiga tiene que hacerse por cada nodo como máximo qv - 1 veces
                  j = 0;
                  do
                  {
                        if(autom == 0)
                        {
                              printf("\n");
                              printf("    Escogeremos el nodo %d\n",
j + 1);
                        }

                        //Debemos repetir este análisis hasta que el
nodo haya sido reconocido como válido
                        accepted = 0;
                        checkIfStuck = 0;
                        do
                        {
                              //A partir del nodo otorgado, vamos al
renglón en vision con el que rellenaremos prob y procedemos a rellenar
prob
                              prob[0][0] = vision[(int)tabu[i][j] -
1][0];
                              prob[0][1] = prob[0][0];
                              for(k = 1; k < qv; k++)
                              {
                                    prob[k][0]                    =
vision[(int)tabu[i][j] - 1][k];
                                    prob[k][1] = prob[k][0] + prob[k
- 1][1];
                              }

                              //Generamos un número aleatorio y
discretizamos el valor entre 10000 posibilidades, todas dentro del rango
                              r = rand() % 10000;
                              r = r/(10000/prob[qv - 1][1]);

                              if(autom == 0)
                              {
                                    printf("            Tenemos las
probabilidades individuales y acumuladas de que de %d se vaya
a\n",(int)tabu[i][j]);
                                    for(ip = 0; ip < qv; ip++)
                                    {
                                          printf("               %d:
", ip + 1);
                                          for(jp = 0; jp < 2; jp++)
```

```
                                        {
                                              printf("[%.4f]    ",
prob[ip][jp]);
                                        }
                                        printf("\n");
                                  }
                                  printf("\n");
                            }

                            //Ubicaremos el nodo seleccionado
                            for(k = 0; k < qv; k++)
                            {
                                  if(r < prob[k][1])
                                  {
                                        //Encontramos que cayó en k
+ 1
                                        break;
                                  }
                            }

                            if(autom == 0)
                            {
                                  printf("       Hemos escogido el
nodo %d (r = %.4f)\n",k + 1,r);
                            }

                            //Ahora toca validar si ese men está
disponible o no
                            accepted                              =
acceptVertex(i,qh,qv,k+1,tabu);

                            if(autom == 0)
                            {
                                  if(accepted == 1)
                                  {
                                        printf("      Revisando el
nodo %d con tabú, obtuvimos que éste es aceptado\n", k + 1);
                                  }
                                  else
                                  {
                                        printf("      Revisando el
nodo %d con tabú, obtuvimos que éste no es aceptado\n", k + 1);
                                  }
                            }

                            //También tenemos que validar si la
hormiga no se quedó encerrada
                            if(accepted == 0)
                            {
                                  checkIfStuck++;
```

```
                                        if(autom == 0)
                                        {
                                              printf("     Tendremos que
repetir la selección aleatoria por posible encrucijada (La hormiga lleva
%d repeticiones)\n", checkIfStuck);
                                        }
                                  }

                                  if(checkIfStuck == 100)
                                  {
                                        if(autom == 0)
                                        {
                                              printf("             Hemos
interpretado que la hormiga se quedó atorada\n Reiniciaremos el proceso
completo\n");
                                        }


      startProcess(i,qh,qv,tabu,beginEnd,availableV);
                                        checkIfStuck = 0;
                                        j = 0;
                                  }
                            }
                            while(accepted == 0);

                            //Hemos aceptado el nodo, vamos a actualizar
la lista tabú y availableV
                            j++;
                            tabu[i][j] = k + 1;
                            availableV[(int)tabu[i][j] - 1] = 0;

                            if(autom == 0)
                            {
                                  printf("       Hemos aceptado el nodo.
Ahora tenemos el camino siguiente\n      ");
                                  for(ip = 0; ip < qh; ip++)
                                  {
                                        for(jp = 0; jp < qv; jp++)
                                        {
                                              printf("[%d]            ",
(int)tabu[ip][jp]);
                                        }
                                        printf("\n      ");
                                  }
                                  printf("\n");
                                  for(ip = 0; ip < qv; ip++)
                                  {
                                        printf("[%d]                  ",
(int)availableV[ip]);
```

```
                                    }
                                    printf("\n");
                              }

                              //Chequemos si el nodo aceptado es el
definido como el último. Así la hormiga se sentirá realizada
                              /*
                              if(k + 1 == beginEnd[i][1])
                              {
                                    if(autom == 0)
                                    {
                                          printf("       Hemos llegado al
nodo objetivo\n");
                                    }
                                    break;
                              }
                              */
                        }
                        while(j < qv - 1);

                        L = getL(i,qh,qv,graph,tabu,beginEnd);

                        if(autom == 0)
                        {
                              printf("\n");
                              printf("    La hormiga produjo un recorrido
de L = %.4f\n",L);
                              getchar();
                        }

                        if(autom != 0)
                        {
                              printf("En la iteración %d, la hormiga %d
produjo el camino:\n",h + 1, i + 1);
                              for(jp = 0; jp < qv; jp++)
                              {
                                    printf("[%d] ", (int)tabu[ip][jp]);
                              }
                              printf("\n");
                              printf("Con un recorrido de valor %.4f\n",L);
                        }

                        for(j = 0; j < qv; j++)
                        {
                              if((j   !=   0)   &&   ((int)tabu[i][j]   ==
(int)beginEnd[i][1]))
                              {
                                    dtau[i][(int)tabu[i][j]             -
1][(int)tabu[i][0] - 1] = Q/L;
                                    break;
```

```
                              }
                              else
                              {
                                    dtau[i][(int)tabu[i][j]              -
1][(int)tabu[i][j + 1] - 1] = Q/L;
                              }
                        }

                        if(autom == 0)
                        {
                              printf("      Se ha generado el siguiente
incremento de feromonas:\n    ");
                              for(jp = 0; jp < qv; jp++)
                              {
                                    for(kp = 0; kp < qv; kp++)
                                    {
                                          printf("[%.4f]                ",
dtau[i][jp][kp]);
                                    }
                                    printf("\n    ");
                              }
                              getchar();
                        }

                        i++;
                  }
                  while(i < qh);

                  //Las  hormigas  han  llegado  al  nodo  destino,  ahora
procedemos a actualizar pheromone
                  for(j = 0; j < qv; j++)
                  {
                        for(k = 0; k < qv; k++)
                        {
                              for(i = 0; i < qh; i++)
                              {
                                    dtau[0][j][k]   =   dtau[0][j][k]   +
dtau[i][j][k];
                              }
                              pheromone[j][k]            =            ((1-
rho)*(pheromone[j][k])) + dtau[0][j][k];
                        }
                  }

                  if(autom == 0)
                  {
                        printf("     Tras el incremento y evaporación de
feromonas, la nueva matriz de feromonas es:\n    ");
                        for(jp = 0; jp < qv; jp++)
                        {
```

```
                        for(kp = 0; kp < qv; kp++)
                        {
                              printf("[%.4f] ", pheromone[jp][kp]);
                        }
                        printf("\n    ");
                  }
                  getchar();
            }


            //La colonia de hormigas ha finalizado, preguntamos por una nueva iteración o terminar el programa, según sea el caso
            if(autom == 0)
            {
                  printf("¿Desea usar un nuevo grupo de hormigas?\n");
                  printf("0. No\n");
                  printf("1. Sí\n");
                  scanf("%d",&repeat1);
                  if(repeat1 == 0)
                  {
                        repeat2 = 0;
                  }
                  if(repeat1 == 1)
                  {
                        repeat2 = 1;
                  }
                  h++;
            }
            else
            {
                  h++;
                  if(h == autom)
                  {
                        repeat2 = 0;
                  }
            }
      }
      while(repeat2 == 1);

      printf("=============== Reporte Final ===============\n");
      printf("Hemos analizado el grafo:\n");
      for(ip = 0; ip < qv; ip++)
      {
            for(jp = 0; jp < qv; jp++)
            {
                  printf("[%d] ", (int)graph[ip][jp]);
            }
            printf("\n");
```

```
            }
            printf("\n");

            printf("La matriz de feromonas resultante es:\n");
            for(ip = 0; ip < qv; ip++)
            {
                  for(jp = 0; jp < qv; jp++)
                  {
                        printf("[%.4f] ", pheromone[ip][jp]);
                  }
                  printf("\n");
            }
            printf("\n");

            printf("La matriz de visibilidad resultante es:\n");
            for(ip = 0; ip < qv; ip++)
            {
                  for(jp = 0; jp < qv; jp++)
                  {
                        printf("[%.4f] ", vision[ip][jp]);
                  }
                  printf("\n");
            }
            printf("\n");




            printf("\n");
            printf("\n");
            printf("\n");
            printf("¿Desea repetir el código?\n");
            printf("0. No\n");
            printf("1. Sí\n");
            scanf("%d",&repeat);
      }
      while(repeat == 1);

      getchar();
}

void setValues1(float alpha, int autom, float beta, int instance, float Q,
int qh, int qv, float rho, float taui,float *alphaP, int *automP, float
*betaP, int *instanceP, float *QP, int *qhP, int *qvP, float *rhoP, float
*tauiP)
{
      int aux, done = 0;
      *alphaP = alpha;
      *automP = autom;
      *betaP = beta;
```

```
	*instanceP = instance;
	*QP = Q;
	*qhP = qh;
	*qvP = qv;
	*rhoP = rho;
	*tauiP = taui;

	//Cargamos la información necesaria de la instancia gr21.tsp
	if(instance == 1)
	{
		qv = 21;
		*qvP = qv;
	}

	//Cargamos la información necesaria de la instancia kroD100.tsp
	if(instance == 2)
	{
		qv = 48;
		*qvP = qv;
	}

	do
	{
		printf("\n");
		printf("--------------------------------\n");
		printf("|  Calibración del algoritmo:\n");
		printf("|  Seleccione algún número si desea cambiar su valor
(o 0 para continuar):\n");
		printf("|  0: Listo. Continuar\n");
		printf("|  1: taui (Tau inicial): %.4f\n",taui);
		printf("|  2: alpha (a): %.4f\n",alpha);
		printf("|  3: beta (a): %.4f\n",beta);
		printf("|  4: rho (a): %.4f\n",rho);
		printf("|  5: Q (a): %.4f\n",Q);
		printf("|  6: qh (Cantidad de hormigas): %d\n",qh);
		printf("|  7: autom (Modo de acción): %d\n",autom);
		if(instance == 0)
		{
			printf("|   8: qv (Cantidad de vértices que posee el
grafo): %d\n",qv);
		}
		printf("|  ");
		scanf("%d",&aux);

		switch (aux)
		{
			case 0:
				{
					//Se continúa con el programa
					done = 1;
```

```
                              break;
                        }
                  case 1:
                        {
                              //Se modifica taui

                              printf("|      Inserte un nuevo valor para
taui (Antiguo valor para taui: T0 = %.4f)\n",taui);
                              printf("|  ");
                              scanf("%f",&taui);
                              *tauiP = taui;

                              break;
                        }
                  case 2:
                        {
                              //Se modifica alpha

                              printf("|      Inserte un nuevo valor para
alpha (Antiguo valor para alpha: alpha = %.4f)\n",alpha);
                              printf("|  ");
                              scanf("%f",&alpha);
                              *alphaP = alpha;

                              break;
                        }
                  case 3:
                        {
                              //Se modifica beta

                              printf("|      Inserte un nuevo valor para
alpha (Antiguo valor para beta: beta = %.4f)\n",beta);
                              printf("|  ");
                              scanf("%f",&beta);
                              *betaP = beta;

                              break;
                        }
                  case 4:
                        {
                              //Se modifica rho

                              printf("|    Inserte un nuevo valor para rho
(Antiguo valor para rho: rho = %.4f)\n",rho);
                              printf("|  ");
                              scanf("%f",&rho);
                              *rhoP = rho;

                              break;
                        }
```

```
                case 5:
                        {
                                //Se modifica Q

                                printf("|     Inserte un nuevo valor para K
(Antiguo valor para Q: Q = %.4f)\n",Q);
                                printf("|  ");
                                scanf("%f",&Q);
                                *QP = Q;

                                break;
                        }
                case 6:
                        {
                                //Se modifica qh

                                printf("|     Inserte un nuevo valor para qh
(Antiguo valor para qh: qh = %d)\n",qh);
                                printf("|  ");
                                scanf("%d",&qh);
                                *qhP = qh;

                                break;
                        }
                case 7:
                        {
                                //Se modifica autom

                                printf("|      Inserte un nuevo valor para
autom (Antiguo valor para autom: autom = %d)\n",autom);
                                printf("|  ");
                                scanf("%d",&autom);
                                *automP = autom;

                                break;
                        }
                case 8:
                        {
                                //Se modifica qv

                                if(instance == 0)
                                {
                                        printf("|      Inserte un nuevo valor
para qv (Antiguo valor para qv: qv = %d)\n",qv);
                                        printf("|  ");
                                        scanf("%d",&qv);
                                }
                                *qvP = qv;

                                break;
```

```
                        }
                default:
                        {
                                //Valor no válido

                                printf("|          Ha  insertado  un  número
inválido\n");

                                break;
                        }
                }
        }
        while(done == 0);

        return;
}

void fillMatrix1D(int a, float matrix[a], float n)
{
        int i,j;
        for(i = 0; i < a; i++)
        {
                matrix[i] = n;
        }
        return;
}

void fillMatrix2D(int a, int b, float matrix[a][b], float n)
{
        int i,j;
        for(i = 0; i < a; i++)
        {
                for(j = 0; j < b; j++)
                {
                        matrix[i][j] = n;
                }
        }
        return;
}

void fillMatrix3D(int a, int b, int c, float matrix[a][b][c], float n)
{
        int i,j,k;
        for(i = 0; i < a; i++)
        {
                for(j = 0; j < b; j++)
                {
                        for(k = 0; k < c; k++)
                        {
                                matrix[i][j][k] = n;
```

```
                }
            }
        }
        return;
}

//Prepara el programa con las configuraciones necesarias y a graph con los valores que se obtengan del archivo gr21.tsp
void setInstance21(int qv, float graph[qv][qv])
{
        //1. Declaramos las variables que usaremos para leer el archivo
        /*
            filetxt: Variable FILE de tipo apuntador a un archivo
            h: Iterador general
            i: Iterador general
            j: Iterador general
            k: Iterador general
            line: Vector que extrae una línea del archivo
            n: Auxiliar para guardar de uno en uno los números que se obtengan del archivo
            reachedgraph: Detecta la línea en la que la matriz comienza (usado en las funciones setInstance21 y setInstance100)
        */
        int h = 0,i,j,k = 0, reachedgraph = 0;
        float n;
        FILE *filetxt;
        char line[200];

        //2. Abrimos el archivo
        filetxt = fopen("gr21.tsp.txt","r");

        //3. Procesamos el archivo y en caso de error regresamos notificándolo al usuario
        if(filetxt == NULL)
        {
            printf("Problemas para abrir el archivo");
            return;
        }

        //4. Leemos el archivo hasta llegar al EOF (End Of File)
        h = 0;
        j = 0;
        k = 1;
        while(feof(filetxt) == 0)
        {
            if(reachedgraph == 0)
            {
                fscanf(filetxt,"%s",line);

                //Encontramos la línea anterior al comienzo del grafo
```

```
                if(strcmp(line, "EDGE_WEIGHT_SECTION") == 0)
                {
                        //Marcamos entonces que hemos alcanzado el grafo y
comenzaremos a obtener su información
                        reachedgraph = 1;
                }
        }
        else
        {
                //Comenzamos a obtener la información del grafo
                if(h != 21)
                {
                        fscanf(filetxt,"%f",&n);
                        graph[h][j] = n;
                        graph[j][h] = n;

                        if(j == k - 1)
                        {
                                h++;
                                j = 0;
                                k++;
                        }
                        else
                        {
                                j++;
                        }
                }
                else
                {
                        fscanf(filetxt,"%s",line);
                }
        }
    }

    //Imprimimos lo que obtuvimos de la lectura de archivos
    printf("El grafo leído del archivo es el siguiente:\n");
    for(i = 0; i < 21; i++)
    {
        for(j = 0; j < 21; j++)
        {
                printf("[%d] ",(int)graph[i][j]);
        }
        printf("\n");
    }

    fclose(filetxt);

    return;
}
```

```
//Prepara el programa con las configuraciones necesarias y a graph con los valores que se obtengan del archivo kroD100.tsp
void setInstance100(int qv, float graph[qv][qv])
{
	//1. Declaramos las variables que usaremos para leer el archivo
	/*
		filetxt: Variable FILE de tipo apuntador a un archivo
		h: Iterador general
		i: Iterador general
		j: Iterador general
		k: Iterador general
		line: Vector que extrae una línea del archivo
		list: Matriz auxiliar donde se guardan en un primer momento los datos previo al cálculo de las distancias entre nodos
		n: Auxiliar para guardar de uno en uno los números que se obtengan del archivo
		reachedgraph: Detecta la línea en la que la matriz comienza (usado en las funciones setInstance21 y setInstance100)
	*/
	int h = 0, i, j, k = 0, list[48][3], n, reachedgraph = 0;
	FILE *filetxt;
	char line[200];

	//2. Abrimos el archivo
	filetxt = fopen("att48.vrp.txt","r");

	//3. Procesamos el archivo y en caso de error regresamos notificándolo al usuario
	if(filetxt == NULL)
	{
		printf("Problemas para abrir el archivo");
		return ;
	}

	//4. Leemos el archivo hasta llegar al EOF (End Of File)
	h = 0;
	j = 0;
	k = 1;
	while(feof(filetxt) == 0)
	{
		if(reachedgraph == 0)
		{
			fscanf(filetxt,"%s",line);

			//Encontramos la línea anterior al comienzo del grafo
			if(strcmp(line, "NODE_COORD_SECTION") == 0)
			{
				//Marcamos entonces que hemos alcanzado el grafo y comenzaremos a obtener su información
				reachedgraph = 1;
```

```
                }
            }
            else
            {
                //Comenzamos a obtener la información del grafo
                if(h != 48)
                {
                    fscanf(filetxt,"%d",&n);
                    list[h][j] = n;
                    j++;

                    if(j == 3)
                    {
                        j = 0;
                        h++;
                    }
                }
                else
                {
                    fscanf(filetxt,"%s",line);
                }
            }
        }

        //Imprimimos lo que obtuvimos de la lectura de archivos
        printf("La lista leída del archivo es la siguiente:\n");
        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 3; j++)
            {
                printf("[%d] ",list[i][j]);
            }
            printf("\n");
        }
        printf("\n");

        //Calculamos las distancias entre los nodos y éstas serán guardadas
en graph
        for(i = 0; i < 48; i++)
        {
            for(j = 0; j < 48; j++)
            {
                graph[i][j]       =       sqrt((pow((list[i][1]       -
list[j][1]),2))+(pow((list[i][2] - list[j][2]),2)));
            }
        }

        //Imprimimos el grafo con el que trabajaremos
        printf("Procesando las distancias, el grafo obtenido es el
siguiente:\n");
```

```
	for(i = 0; i < 48; i++)
	{
		for(j = 0; j < 48; j++)
		{
			printf("[%.4f] ",graph[i][j]);
		}
		printf("\n");
	}

	fclose(filetxt);
}

void setValues2(int qv, int qh, float graph[qv][qv], float pheromone[qv][qv], float vision[qv][qv], float beginEnd[qh][2])
{
	int aux, done = 0,i,j;
	do
	{
		printf("\n");
		printf("--------------------------------\n");
		printf("|  Calibración del algoritmo:\n");
		printf("|  Seleccione algún número si desea cambiar su valor (o 0 para continuar):\n");
		printf("|  0: Listo. Continuar\n");
		printf("|  1: graph (El grafo a evaluar):\n");
		for(i = 0; i < qv; i++)
		{
			printf("|     ");
			for(j = 0; j < qv; j++)
			{
				printf("[%d] ",(int)graph[i][j]);
			}
			printf("\n");
		}
		printf("|  2: BeginEnd (Los puntos de inicio y final de cada hormiga):\n");
		for(i = 0; i < qh; i++)
		{
			printf("|     ");
			for(j = 0; j < 2; j++)
			{
				printf("[%d] ",(int)beginEnd[i][j]);
			}
			printf("\n");
		}
		printf("|  ");
		scanf("%d",&aux);

		switch (aux)
		{
```

```
                  case 0:
                        {
                              //Se continúa con el programa
                              done = 1;
                              break;
                        }
                  case 1:
                        {
                              //Se modifica graph

                              for(i = 0; i < qv; i++)
                              {
                                    for(j = 0; j < qv; j++)
                                    {
                                          printf("|     Inserte el peso de
la arista que une al vértice %d con el vértice %d (0 para indicar que no
existe tal unión)\n",i + 1, j + 1);
                                          printf("|  ");
                                          scanf("%f",&graph[i][j]);
                                    }
                              }

                              break;
                        }
                  case 2:
                        {
                              //Se modifican los puntos de arranque y fin
de cada hormiga (tabu)

                              for(i = 0; i < qh; i++)
                              {
                                    printf("|     Inserte el nodo inicial
y final de la hormiga %d:\n",i + 1);
                                    printf("|  ");
                                    scanf("%f",&beginEnd[i][0]);
                                    beginEnd[i][1] = beginEnd[i][0];
                              }

                              break;
                        }
            }
      }
      while(done == 0);
}

void  startProcess(int  i,  int  qh,  int  qv,  float  tabu[qh][qv],  float
beginEnd[qh][2], float availableV[qv])
{
      int j;
```

```
	for(j = 0; j < qv; j++)
	{
		tabu[i][j] = 0;
		availableV[j] = 1;
	}

	tabu[i][0] = (int)beginEnd[i][0];
	tabu[i][qv] = (int)beginEnd[i][0];
	availableV[(int)tabu[i][0] - 1] = 0;

	return;
}

int acceptVertex(int i, int qh, int qv, int value, float tabu[qh][qv])
{
	int j;
	for(j = 0; j < qv; j++)
	{
		if((int)tabu[i][j] == value)
		{
			return 0;
		}
	}
	return 1;
}

float getL(int i, int qh, int qv, float graph[qv][qv], float tabu[qh][qv],
float beginEnd[qh][2])
{
	int j;
	float L = 0;
	for(j = 0; j < qv; j++)
	{
		if((j != 0) && ((int)tabu[i][j] == (int)beginEnd[i][1]))
		{
			L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][0] - 1];
			return L;
		}
		else
		{
			L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][j + 1] -
1];
		}
	}
}
```

Anexo 3: Código de solución de la Colonia de Hormigas para el Problema de Ruteo de Vehículos en Paralelo.

```
/*
    Universidad Autónoma de Aguascalientes

          Centro de Ciencias Básicas
  Departamento de Ciencias de la Computación
         Paralelización de Algoritmos

                    9° "A"

            Tercera Evaluación Parcial
               Entrega del Proyecto

        Dr. Julio César Ponce Gallegos

                  Alumnos:
        Adame Michelle Stephanie
        Espinoza Sánchez Joel Alejandro
        Gómez Garza Dariana
        González Arenas Fernando Francisco

  Fecha de Entrega: 8 de diciembre del 2022
*/

//Cargamos las librerías
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <time.h>
#include <math.h>
#include <string.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

cudaError_t nodeDistanceCuda(int *graph, int *list, int num);
cudaError_t visionInitCuda(float *vision, float *pheromone, int *graph,
float alpha, float beta, int qv);

//void setInstance21(int qv, float graph[qv][qv]);

__global__ void addKernelnodeDistanceCuda(int *graph, int *list)
{
    int i = threadIdx.x;
    int j = threadIdx.x;
    graph[i][j] = sqrt((pow((list[i][1] - list[j][1]),2))+(pow((list[i][2]
- list[j][2]),2)));
}
```

```
__global__ void addKernelvisionInitCuda(float *vision, float *pheromone,
int *graph, float alpha, float beta)
{
    int i = threadIdx.x;
    int j = threadIdx.x;
    if(graph[i][j] == 0)
    {
        vision[i][j] = 0;
    }
    else
    {
        vision[i][j]                                                    =
(pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
    }
}

main()
{
      setlocale(LC_ALL,"");
      srand(time(NULL));

      //1. Declaramos las variables que usaremos
      /*
            qv: Cantidad de vértices del grafo
            qh: Cantidad de hormigas
            tau: Valor de la feromona inicial
            Q: Parámetro
            alpha: Parámetro
            beta: Parámetro
            instance: Instancia que se usará en el recocido, donde:
                        Si   instance   =   0   ejecutará   un   proceso
personalizado
                        Si  instance  =  1  ejecutará  el  proceso  de  la
instancia pequeña
                        Si  instance  =  2  ejecutará  el  proceso  de  la
instancia grande
            rho: Parámetro
            autom: Modo de acción (Para autom >=1 se tomará el número de
autom como las repeticiones)
            repeat: Sirve para repetir todo el código nuevamente
            repeat1: Sirve para repetir la colonia en modo manual
            r: Número aleatorio
            accepted: Bandera de aceptación de vértice
            checkIfStuck: Iterador que no tolerará un número determinado
de iteraciones. Si se alcanzan, se interpretará que la hormiga se encerrá
            L: Longitud del recorrido completo realizado por la hormiga
      */
      int qv, qh, autom, instance, repeat, repeat1, repeat2, accepted,
checkIfStuck, h, i, j, k, ip, jp, kp, done;
```

```
      float tau, Q, alpha, beta, rho, L, r;

      printf("============================    COLONIA    DE    HORMIGAS
============================\n");

      do
      {
            alpha = 1;
            autom = 8;
            beta = 1;
            Q = 1;
            qh = 1;
            qv = 7;
            repeat = 0;
            rho = 0.01;
            tau = 0.1;

            //3. Calibramos este primer conjunto de valores del algoritmo
            printf("\n");
            printf("--------------------------------\n");
            printf("|  Calibración del algoritmo:\n");
            printf("|  Seleccione la instancia a trabajar:\n");
            printf("|    0: Grafo Personalizado\n");
            printf("|    1: Instancia pequeña (gr21.tsp)\n");
            printf("|    2: Instancia grande (kroD100.tsp)\n");
            printf("|  ");
            scanf("%d",&instance);

            //2. Calibramos los primeros valores del programa

      setValues1(alpha,autom,beta,instance,Q,qh,qv,rho,tau,&alpha,&autom,
&beta,&instance,&Q,&qh,&qv,&rho,&tau);
        //Cargamos la información necesaria de la instancia gr21.tsp
        if(instance == 1)
        {
            qv = 21;
            *qvP = qv;
        }

        //Cargamos la información necesaria de la instancia kroD100.tsp
        if(instance == 2)
        {
            qv = 48;
            *qvP = qv;
        }

        do
        {
            printf("\n");
            printf("--------------------------------\n");
```

```
            printf("|  Calibración del algoritmo:\n");
            printf("|  Seleccione algún número si desea cambiar su valor
(o 0 para continuar):\n");
            printf("|  0: Listo. Continuar\n");
            printf("|  1: taui (Tau inicial): %.4f\n",taui);
            printf("|  2: alpha (a): %.4f\n",alpha);
            printf("|  3: beta (a): %.4f\n",beta);
            printf("|  4: rho (a): %.4f\n",rho);
            printf("|  5: Q (a): %.4f\n",Q);
            printf("|  6: qh (Cantidad de hormigas): %d\n",qh);
            printf("|  7: autom (Modo de acción): %d\n",autom);
            if(instance == 0)
            {
                printf("|  8: qv (Cantidad de vértices que posee el grafo):
%d\n",qv);
            }
            printf("|  ");
            scanf("%d",&aux);

            switch (aux)
            {
                case 0:
                    {
                        //Se continúa con el programa
                        done = 1;
                        break;
                    }
                case 1:
                    {
                        //Se modifica taui

                        printf("|      Inserte un nuevo valor para taui
(Antiguo valor para taui: T0 = %.4f)\n",taui);
                        printf("|  ");
                        scanf("%f",&taui);
                        *tauiP = taui;

                        break;
                    }
                case 2:
                    {
                        //Se modifica alpha

                        printf("|      Inserte un nuevo valor para alpha
(Antiguo valor para alpha: alpha = %.4f)\n",alpha);
                        printf("|  ");
                        scanf("%f",&alpha);
                        *alphaP = alpha;

                        break;
```

```
                }
            case 3:
                {
                    //Se modifica beta

                    printf("|      Inserte un nuevo valor para alpha
(Antiguo valor para beta: beta = %.4f)\n",beta);
                    printf("|  ");
                    scanf("%f",&beta);
                    *betaP = beta;

                    break;
                }
            case 4:
                {
                    //Se modifica rho

                    printf("|       Inserte un nuevo valor para rho
(Antiguo valor para rho: rho = %.4f)\n",rho);
                    printf("|  ");
                    scanf("%f",&rho);
                    *rhoP = rho;

                    break;
                }
            case 5:
                {
                    //Se modifica Q

                    printf("|        Inserte un nuevo valor para K
(Antiguo valor para Q: Q = %.4f)\n",Q);
                    printf("|  ");
                    scanf("%f",&Q);
                    *QP = Q;

                    break;
                }
            case 6:
                {
                    //Se modifica qh

                    printf("|       Inserte un nuevo valor para qh
(Antiguo valor para qh: qh = %d)\n",qh);
                    printf("|  ");
                    scanf("%d",&qh);
                    *qhP = qh;

                    break;
                }
            case 7:
```

```
                    {
                        //Se modifica autom

                        printf("|      Inserte un nuevo valor para autom
(Antiguo valor para autom: autom = %d)\n",autom);
                        printf("|  ");
                        scanf("%d",&autom);
                        *automP = autom;

                        break;
                    }
                case 8:
                    {
                        //Se modifica qv

                        if(instance == 0)
                        {
                            printf("|      Inserte un nuevo valor para qv
(Antiguo valor para qv: qv = %d)\n",qv);
                            printf("|  ");
                            scanf("%d",&qv);
                        }
                        *qvP = qv;

                        break;
                    }
                default:
                    {
                        //Valor no válido

                        printf("|    Ha insertado un número inválido\n");

                        break;
                    }
            }
        }
        while(done == 0);

            //3. Declaramos más variables
            /*
                  graph: La matriz de adyacencia del grafo que analizaremos

            INT
                  pheromone: La matriz de feromonas

                        FLOAT
                  vision: La matriz de visibilidad

                        FLOAT
```

```
                  tabu: La lista tabú

                                    INT
                  beginEnd: Matriz con los valores de inicio y fin de cada
hormiga
      INT
                  availableV: Vector donde cada elemento representa cada
vértice del grafo (1 está disponible o 0 no lo está)      INT
                  prob: Probabilidad individual y acumulada de tomar cada
nodo
      FLOAT
                  dtau: La diferencia de tau que hay por hormiga en cada
nodo
            FLOAT
            */
            float   graph[qv][qv],   pheromone[qv][qv],   vision[qv][qv],
tabu[qh][qv],     beginEnd[qh][2],     availableV[qv],     prob[qv][2],
dtau[qh][qv][qv];

            //5. Calibramos este segundo conjunto de valores del algoritmo
            if(instance == 1)
            {
                  //setInstance21(qv,graph);
            }
            if(instance == 2)
            {
                  //1. Declaramos las variables que usaremos para leer el
archivo
            /*
                filetxt: Variable FILE de tipo apuntador a un archivo
                h: Iterador general
                i: Iterador general
                j: Iterador general
                k: Iterador general
                line: Vector que extrae una línea del archivo
                list: Matriz auxiliar donde se guardan en un primer momento
los datos previo al cálculo de las distancias entre nodos
                n: Auxiliar para guardar de uno en uno los números que se
obtengan del archivo
                reachedgraph: Detecta la línea en la que la matriz comienza
(usado en las funciones setInstance21 y setInstance100)
            */
            int h = 0, i, j, k = 0, list[48][3], n, reachedgraph = 0;
            FILE *filetxt;
            char line[200];

            //2. Abrimos el archivo
            filetxt = fopen("att48.vrp.txt","r");
```

```
            //3. Procesamos el archivo y en caso de error regresamos
notificándolo al usuario
            if(filetxt == NULL)
            {
                printf("Problemas para abrir el archivo");
                return ;
            }

            //4. Leemos el archivo hasta llegar al EOF (End Of File)
            h = 0;
            j = 0;
            k = 1;
            while(feof(filetxt) == 0)
            {
                if(reachedgraph == 0)
                {
                    fscanf(filetxt,"%s",line);

                    //Encontramos la línea anterior al comienzo del grafo
                    if(strcmp(line, "NODE_COORD_SECTION") == 0)
                    {
                        //Marcamos entonces que hemos alcanzado el grafo
y comenzaremos a obtener su información
                        reachedgraph = 1;
                    }
                }
                else
                {
                    //Comenzamos a obtener la información del grafo
                    if(h != 48)
                    {
                        fscanf(filetxt,"%d",&n);
                        list[h][j] = n;
                        j++;

                        if(j == 3)
                        {
                            j = 0;
                            h++;
                        }
                    }
                    else
                    {
                        fscanf(filetxt,"%s",line);
                    }
                }
            }

            //Imprimimos lo que obtuvimos de la lectura de archivos
            printf("La lista leída del archivo es la siguiente:\n");
```

```
            for(i = 0; i < 48; i++)
            {
                for(j = 0; j < 3; j++)
                {
                    printf("[%d] ",list[i][j]);
                }
                printf("\n");
            }
            printf("\n");

            //Calculamos las distancias entre los nodos y éstas serán
guardadas en graph
            cudaError_t cudaStatus = nodeDistanceCuda(graph, list, 48);
            if (cudaStatus != cudaSuccess) {
                fprintf(stderr, "nodeDistanceCuda failed!");
                return 1;
            }

            /*
            for(i = 0; i < 48; i++)
            {
                for(j = 0; j < 48; j++)
                {
                    graph[i][j]      =      sqrt((pow((list[i][1]      -
list[j][1]),2))+(pow((list[i][2] - list[j][2]),2)));
                }
            }
            */

            //Imprimimos el grafo con el que trabajaremos
            printf("Procesando las distancias, el grafo obtenido es el
siguiente:\n");
            for(i = 0; i < 48; i++)
            {
                for(j = 0; j < 48; j++)
                {
                    printf("[%.4f] ",graph[i][j]);
                }
                printf("\n");
            }

            fclose(filetxt);
            }

            //4. Inicializamos las estructuras
            if(instance == 0)
            {
                  //Limpiamos graph
                  for(i = 0; i < qv; i++)
            {
```

```
        for(j = 0; j < qv; j++)
        {
            graph[i][j] = 0;
        }
    }
    }

    //Inicializamos pheromone
    for(i = 0; i < qv; i++)
{
    for(j = 0; j < qv; j++)
    {
        pheromone[i][j] = tau;
    }
}

    //Limpiamos vision
    for(i = 0; i < qv; i++)
{
    for(j = 0; j < qv; j++)
    {
        vision[i][j] = 0;
    }
}

    //Limpiamos tabu
    for(i = 0; i < qh; i++)
{
    for(j = 0; j < qv; j++)
    {
        tabu[i][j] = 0;
    }
}

    //Limpiamos beginEnd
    for(i = 0; i < qh; i++)
{
    for(j = 0; j < 2; j++)
    {
        beginEnd[i][j] = 0;
    }
}

    //Inicializamos availableV
    for(i = 0; i < qv; i++)
{
    availableV[i] = 1;
}

    //Limpiamos prob
```

```
            for(i = 0; i < qv; i++)
        {
            for(j = 0; j < 2; j++)
            {
                prob[i][j] = 0;
            }
        }

            //5. Calibramos los demás valores del programa
            int aux, done = 0;
        do
        {
            printf("\n");
            printf("--------------------------------\n");
            printf("|  Calibración del algoritmo:\n");
            printf("|  Seleccione algún número si desea cambiar su valor
(o 0 para continuar):\n");
            printf("|  0: Listo. Continuar\n");
            printf("|  1: graph (El grafo a evaluar):\n");
            for(i = 0; i < qv; i++)
            {
                printf("|     ");
                for(j = 0; j < qv; j++)
                {
                    printf("[%d] ",(int)graph[i][j]);
                }
                printf("\n");
            }
            printf("|  2: BeginEnd (Los puntos de inicio y final de cada
hormiga):\n");
            for(i = 0; i < qh; i++)
            {
                printf("|     ");
                for(j = 0; j < 2; j++)
                {
                    printf("[%d] ",(int)beginEnd[i][j]);
                }
                printf("\n");
            }
            printf("|  ");
            scanf("%d",&aux);

            switch (aux)
            {
                case 0:
                    {
                        //Se continúa con el programa
                        done = 1;
                        break;
                    }
```

```
                case 1:
                    {
                        //Se modifica graph

                        for(i = 0; i < qv; i++)
                        {
                            for(j = 0; j < qv; j++)
                            {
                                printf("|    Inserte el peso de la arista
que une al vértice %d con el vértice %d (0 para indicar que no existe tal
unión)\n",i + 1, j + 1);
                                printf("|  ");
                                scanf("%f",&graph[i][j]);
                            }
                        }

                        break;
                    }
                case 2:
                    {
                        //Se modifican los puntos de arranque y fin de
cada hormiga (tabu)

                        for(i = 0; i < qh; i++)
                        {
                            printf("|     Inserte el nodo inicial y final
de la hormiga %d:\n",i + 1);
                            printf("|  ");
                            scanf("%f",&beginEnd[i][0]);
                            beginEnd[i][1] = beginEnd[i][0];
                        }

                        break;
                    }
            }
        }
        while(done == 0);

            //Inicializamos vision
        cudaError_t cudaStatus = visionInitCuda(vision, pheromone, graph,
alpha, beta, qv);
        if (cudaStatus != cudaSuccess) {
            fprintf(stderr, "nodeDistanceCuda failed!");
            return 1;
        }

        /*
            for(i = 0; i < qv; i++)
            {
                  for(j = 0; j < qv; j++)
```

```
                {
                        if(graph[i][j] == 0)
                        {
                                vision[i][j] = 0;
                        }
                        else
                        {
                                vision[i][j]                                    =
(pow(pheromone[i][j],alpha))/(pow(graph[i][j],beta));
                        }
                }
            }
        */

            printf("\n\n\n\n");

            if(autom == 0)
            {
                printf("El grafo a evaluar es:\n");
                for(ip = 0; ip < qv; ip++)
                {
                        for(jp = 0; jp < qv; jp++)
                        {
                                printf("[%d] ", (int)graph[ip][jp]);
                        }
                        printf("\n");
                }
                printf("\n");

                printf("La matriz de feromonas antes del comienzo del
algoritmo es:\n");
                for(ip = 0; ip < qv; ip++)
                {
                        for(jp = 0; jp < qv; jp++)
                        {
                                printf("[%.4f] ", pheromone[ip][jp]);
                        }
                        printf("\n");
                }
                printf("\n");

                printf("La matriz de visibilidad, entonces, es:\n");
                for(ip = 0; ip < qv; ip++)
                {
                        for(jp = 0; jp < qv; jp++)
                        {
                                printf("[%.4f] ", vision[ip][jp]);
                        }
                        printf("\n");
                }
```

```
                printf("\n");
                getchar();
                getchar();
        }

        //6. Comenzamos la colonia de hormigas, la cual se repetirá
tantas veces como se haya calibrado autom (si es 0, se decide el paro)
        h = 0;
        repeat2 = 1;
        do
        {
                //Inicializamos dtau
                for(i = 0; i < qh; i++)
        {
            for(j = 0; j < qv; j++)
            {
                for(k = 0; k < qv; k++)
                {
                    dtau[i][j][k] = 0;
                }
            }
        }

                if(autom == 0)
                {
                        printf("\n");
                        printf("\n");
                        printf("Comenzamos la %d iteración\n\n", h + 1);
                }

                //El  procedimiento  de  la  colonia  se  hará  para  cada
hormiga
                i = 0;
                do
                {
                        if(autom == 0)
                        {
                                printf("    Comenzamos  a  trabajar  con  la
hormiga %d\n", i + 1);
                                getchar();
                        }

                        //Inicializamos tabu y availableV con los valores
proporcionados por beginEnd
                        for(j = 0; j < qv; j++)
                {
                    tabu[i][j] = 0;
                    availableV[j] = 1;
                }
```

```
                tabu[i][0] = (int)beginEnd[i][0];
                tabu[i][qv] = (int)beginEnd[i][0];
                availableV[(int)tabu[i][0] - 1] = 0;

                        if(autom == 0)
                        {
                                printf("    Inicializamos la lista tabú con
el valor del nodo de inicio:\n    ");
                                for(ip = 0; ip < qh; ip++)
                                {
                                        for(jp = 0; jp < qv; jp++)
                                        {
                                                printf("[%d]                  ",
(int)tabu[ip][jp]);
                                        }
                                        printf("\n    ");
                                }

                                printf("\n");
                                printf("    Inicializamos la lista de nodos
disponibles:\n    ");
                                for(ip = 0; ip < qv; ip++)
                                {
                                        printf("[%d] ", (int)availableV[ip]);
                                }
                                printf("\n");

                                getchar();
                        }

                        //Todavía, el procedimiento de selección de camino
de cada hormiga tiene que hacerse por cada nodo como máximo qv - 1 veces
                        j = 0;
                        do
                        {
                                if(autom == 0)
                                {
                                        printf("\n");
                                        printf("    Escogeremos el nodo %d\n",
j + 1);
                                }

                                //Debemos repetir este análisis hasta que el
nodo haya sido reconocido como válido
                                accepted = 0;
                                checkIfStuck = 0;
                                do
                                {
```

```
                                    //A partir del nodo otorgado, vamos al
renglón en vision con el que rellenaremos prob y procedemos a rellenar
prob
                                    prob[0][0] = vision[(int)tabu[i][j] -
1][0];
                                    prob[0][1] = prob[0][0];
                                    for(k = 1; k < qv; k++)
                                    {
                                          prob[k][0]                    =
vision[(int)tabu[i][j] - 1][k];
                                          prob[k][1] = prob[k][0] + prob[k
- 1][1];
                                    }

                                    //Generamos un número aleatorio y
discretizamos el valor entre 10000 posibilidades, todas dentro del rango
                                    r = rand() % 10000;
                                    r = r/(10000/prob[qv - 1][1]);

                                    if(autom == 0)
                                    {
                                          printf("            Tenemos las
probabilidades individuales y acumuladas de que de %d se vaya
a\n",(int)tabu[i][j]);
                                          for(ip = 0; ip < qv; ip++)
                                          {
                                                printf("               %d:
", ip + 1);
                                                for(jp = 0; jp < 2; jp++)
                                                {
                                                      printf("[%.4f]    ",
prob[ip][jp]);
                                                }
                                                printf("\n");
                                          }
                                          printf("\n");
                                    }

                                    //Ubicaremos el nodo seleccionado
                                    for(k = 0; k < qv; k++)
                                    {
                                          if(r < prob[k][1])
                                          {
                                                //Encontramos que cayó en k
+ 1
                                                break;
                                          }
                                    }

                                    if(autom == 0)
```

```
                                    {
                                          printf("      Hemos escogido el
nodo %d (r = %.4f)\n",k + 1,r);
                                    }

                                    //Ahora toca validar si ese men está
disponible o no
                                    accepted = 1;
                              for(j = 0; j < qv; j++)
                              {
                                  if((int)tabu[i][j] == value)
                                  {
                                      accepted = 0;
                                  }
                              }

                                    if(autom == 0)
                                    {
                                          if(accepted == 1)
                                          {
                                                printf("      Revisando el
nodo %d con tabú, obtuvimos que éste es aceptado\n", k + 1);
                                          }
                                          else
                                          {
                                                printf("      Revisando el
nodo %d con tabú, obtuvimos que éste no es aceptado\n", k + 1);
                                          }
                                    }

                                    //También tenemos que validar si la
hormiga no se quedó encerrada
                                    if(accepted == 0)
                                    {
                                          checkIfStuck++;

                                          if(autom == 0)
                                          {
                                                printf("     Tendremos que
repetir la selección aleatoria por posible encrucijada (La hormiga lleva
%d repeticiones)\n", checkIfStuck);
                                          }
                                    }

                                    if(checkIfStuck == 100)
                                    {
                                          if(autom == 0)
                                          {
```

```
                                        printf("            Hemos
interpretado que la hormiga se quedó atorada\n Reiniciaremos el proceso
completo\n");
                                        }

                                        for(j = 0; j < qv; j++)
                                {
                                    tabu[i][j] = 0;
                                    availableV[j] = 1;
                                }

                                tabu[i][0] = (int)beginEnd[i][0];
                                tabu[i][qv] = (int)beginEnd[i][0];
                                availableV[(int)tabu[i][0] - 1] = 0;
                                        checkIfStuck = 0;
                                        j = 0;
                                    }
                                 }
                                 while(accepted == 0);

                                 //Hemos aceptado el nodo, vamos a actualizar
la lista tabú y availableV
                                 j++;
                                 tabu[i][j] = k + 1;
                                 availableV[(int)tabu[i][j] - 1] = 0;

                                 if(autom == 0)
                                 {
                                        printf("      Hemos aceptado el nodo.
Ahora tenemos el camino siguiente\n      ");
                                        for(ip = 0; ip < qh; ip++)
                                        {
                                            for(jp = 0; jp < qv; jp++)
                                            {
                                                printf("[%d]            ",
(int)tabu[ip][jp]);
                                            }
                                            printf("\n      ");
                                        }
                                        printf("\n");
                                        for(ip = 0; ip < qv; ip++)
                                        {
                                            printf("[%d]                 ",
(int)availableV[ip]);
                                        }
                                        printf("\n");
                                 }

                                 //Chequemos si el nodo aceptado es el
definido como el último. Así la hormiga se sentirá realizada
```

```
                        /*
                        if(k + 1 == beginEnd[i][1])
                        {
                              if(autom == 0)
                              {
                                    printf("       Hemos llegado al
nodo objetivo\n");
                              }
                              break;
                        }
                        */
                    }
                    while(j < qv - 1);

                    L = 0;
               for(j = 0; j < qv; j++)
               {
                   if((j    !=    0)    &&    ((int)tabu[i][j]    ==
(int)beginEnd[i][1]))
                   {
                       L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][0]
- 1];
                       break;
                   }
                   else
                   {
                       L = L + graph[(int)tabu[i][j] - 1][(int)tabu[i][j
+ 1] - 1];
                   }
               }

                    if(autom == 0)
                    {
                          printf("\n");
                          printf("    La hormiga produjo un recorrido
de L = %.4f\n",L);
                          getchar();
                    }

                    if(autom != 0)
                    {
                          printf("En la iteración %d, la hormiga %d
produjo el camino:\n",h + 1, i + 1);
                          for(jp = 0; jp < qv; jp++)
                          {
                                printf("[%d] ", (int)tabu[ip][jp]);
                          }
                          printf("\n");
                          printf("Con un recorrido de valor %.4f\n",L);
                    }
```

```
                        for(j = 0; j < qv; j++)
                        {
                              if((j   !=   0)   &&   ((int)tabu[i][j]   ==
(int)beginEnd[i][1]))
                              {
                                    dtau[i][(int)tabu[i][j]              -
1][(int)tabu[i][0] - 1] = Q/L;
                                    break;
                              }
                              else
                              {
                                    dtau[i][(int)tabu[i][j]              -
1][(int)tabu[i][j + 1] - 1] = Q/L;
                              }
                        }

                        if(autom == 0)
                        {
                              printf("      Se  ha  generado  el  siguiente
incremento de feromonas:\n    ");
                              for(jp = 0; jp < qv; jp++)
                              {
                                    for(kp = 0; kp < qv; kp++)
                                    {
                                          printf("[%.4f]                ",
dtau[i][jp][kp]);
                                    }
                                    printf("\n    ");
                              }
                              getchar();
                        }

                        i++;
                  }
                  while(i < qh);

                  //Las  hormigas  han  llegado  al  nodo  destino,  ahora
procedemos a actualizar pheromone
                  for(j = 0; j < qv; j++)
                  {
                        for(k = 0; k < qv; k++)
                        {
                              for(i = 0; i < qh; i++)
                              {
                                    dtau[0][j][k]   =   dtau[0][j][k]   +
dtau[i][j][k];
                              }
                              pheromone[j][k]            =            ((1-
rho)*(pheromone[j][k])) + dtau[0][j][k];
```

```
                        }
                  }

                  if(autom == 0)
                  {
                        printf("     Tras el incremento y evaporación de
feromonas, la nueva matriz de feromonas es:\n    ");
                        for(jp = 0; jp < qv; jp++)
                        {
                              for(kp = 0; kp < qv; kp++)
                              {
                                    printf("[%.4f] ", pheromone[jp][kp]);
                              }
                              printf("\n    ");
                        }
                        getchar();
                  }



                  //La colonia de hormigas ha finalizado, preguntamos por
una nueva iteración o terminar el programa, según sea el caso
                  if(autom == 0)
                  {
                        printf("¿Desea   usar   un   nuevo   grupo   de
hormigas?\n");
                        printf("0. No\n");
                        printf("1. Sí\n");
                        scanf("%d",&repeat1);
                        if(repeat1 == 0)
                        {
                              repeat2 = 0;
                        }
                        if(repeat1 == 1)
                        {
                              repeat2 = 1;
                        }
                        h++;
                  }
                  else
                  {
                        h++;
                        if(h == autom)
                        {
                              repeat2 = 0;
                        }
                  }
            }
            while(repeat2 == 1);
```

```
	printf("=============== Reporte Final ===============\n");
	printf("Hemos analizado el grafo:\n");
	for(ip = 0; ip < qv; ip++)
	{
		for(jp = 0; jp < qv; jp++)
		{
			printf("[%d] ", (int)graph[ip][jp]);
		}
		printf("\n");
	}
	printf("\n");

	printf("La matriz de feromonas resultante es:\n");
	for(ip = 0; ip < qv; ip++)
	{
		for(jp = 0; jp < qv; jp++)
		{
			printf("[%.4f] ", pheromone[ip][jp]);
		}
		printf("\n");
	}
	printf("\n");

	printf("La matriz de visibilidad resultante es:\n");
	for(ip = 0; ip < qv; ip++)
	{
		for(jp = 0; jp < qv; jp++)
		{
			printf("[%.4f] ", vision[ip][jp]);
		}
		printf("\n");
	}
	printf("\n");




	printf("\n");
	printf("\n");
	printf("\n");
	printf("¿Desea repetir el código?\n");
	printf("0. No\n");
	printf("1. Sí\n");
	scanf("%d",&repeat);
}
while(repeat == 1);

getchar();
```

```
    // cudaDeviceReset se llama para generar reportes de Nsight y Visual
Profiler
    cudaStatus = cudaDeviceReset();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceReset failed!");
        return 1;
    }
}

cudaError_t nodeDistanceCuda(int *graph, int *list, int num)
{
    int *dev_graph = 0;
    int *dev_list = 0;
    cudaError_t cudaStatus;

    // Se elige la GPU de ejecución
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable
GPU installed?");
        goto Error;
    }

    // Se fijan buffers de GPU para dos arreglos (un input, un output)
    cudaStatus = cudaMalloc((void**)&dev_graph, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_list, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    // Se copia el arreglo de entrada de memoria host a buffer de GPU
    cudaStatus   =   cudaMemcpy(dev_list,   list,   size   *   sizeof(int),
cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

    // Se lanza el kernel en la GPU con un hilo por elemento
    addKernelnodeDistanceCuda<<<1, size>>>(dev_graph, dev_list);

    // Revisión de errores al lanzar el kernel
    cudaStatus = cudaGetLastError();
    if (cudaStatus != cudaSuccess) {
```

```
        fprintf(stderr,     "addKernel     launch     failed:     %s\n",
cudaGetErrorString(cudaStatus));
        goto Error;
    }

    // cudaDeviceSynchronize  espera al kernel y devuelve errores durante
el launch
    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr,  "cudaDeviceSynchronize  returned  error  code  %d
after launching addKernel!\n", cudaStatus);
        goto Error;
    }

    // Se copia el arreglo de buffer de GPU a memoria host
    cudaStatus  =  cudaMemcpy(graph,  dev_graph,  size  *  sizeof(int),
cudaMemcpyDeviceToHost);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

Error:
    cudaFree(dev_graph);
    cudaFree(dev_list);

    return cudaStatus;
}

cudaError_t visionInitCuda(float *vision, float *pheromone, int *graph,
float alpha, float beta, int qv)
{
    int *dev_vision = 0;
    int *dev_pheromone = 0;
    int *dev_graph = 0;
    cudaError_t cudaStatus;

    // Se elige la GPU de ejecución
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable
GPU installed?");
        goto Error;
    }

    // Se fijan buffers de GPU para tres arreglos (dos input, un output)
    cudaStatus = cudaMalloc((void**)&dev_vision, size * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
```

```
    }

    cudaStatus = cudaMalloc((void**)&dev_pheromone, size * sizeof(float));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_graph, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    // Se copian los arreglos de entrada de memoria host a buffer de GPU
    cudaStatus    =    cudaMemcpy(dev_pheromone,    pheromone,    size    *
sizeof(float), cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

    cudaStatus  =  cudaMemcpy(dev_graph,  graph,  size  *  sizeof(int),
cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

    // Se lanza el kernel en la GPU con un hilo por elemento
    addKernelvisionInitCuda<<<1,    size>>>(dev_vision,    dev_pheromone,
dev_graph, alpha, beta);

    // Revisión de errores al lanzar el kernel
    cudaStatus = cudaGetLastError();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr,     "addKernel     launch     failed:     %s\n",
cudaGetErrorString(cudaStatus));
        goto Error;
    }

    // cudaDeviceSynchronize  espera al kernel y devuelve errores durante
el launch
    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr,  "cudaDeviceSynchronize  returned  error  code  %d
after launching addKernel!\n", cudaStatus);
        goto Error;
    }
```

```
    // Se copia el arreglo de buffer de GPU a memoria host
    cudaStatus = cudaMemcpy(graph, dev_graph, size * sizeof(int),
cudaMemcpyDeviceToHost);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

Error:
    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);

    return cudaStatus;
}
```