

3/12/2024

Chen Yangfeng

6BV1 PROF. DANIEL MOLINA PÉREZ

Ruta De Las Metrópolis Con Ventanas De Tiempo (Tsp-Tw)

[Tópicos selectos Bioinspirados](https://classroom.google.com/u/0/c/Njg1MjQyNzg3MDIw)

A map of the united states

Description automatically generated

Índice

[Descripción del problema 1](#_Toc183773869)

[Descripción del algoritmo genético híbrido 2](#_Toc183773870)

[Parámetros usados 3](#_Toc183773871)

[Tabla con los resultados estadísticos 3](#_Toc183773872)

[Resultados obtenidos 5](#_Toc183773873)

[Código 5](#_Toc183773874)

# Descripción del problema

El **Problema del Agente Viajero con Ventanas de Tiempo (TSP-TW)** es una variante del clásico Problema del Agente Viajero (TSP), en el cual un agente debe visitar un conjunto de ciudades exactamente una vez, minimizando la distancia o el costo total del recorrido y regresando al punto de partida. En la variante TSP-TW, cada ciudad tiene asociada una ventana de tiempo durante la cual debe ser visitada. Si el agente llega a una ciudad antes de que la ventana de tiempo esté abierta, debe esperar hasta que sea permitido ingresar. Este problema es un ejemplo de optimización combinatoria con restricciones temporales y es ampliamente utilizado en logística, transporte y planificación de rutas.

En este caso particular, la **"Ruta de las Metrópolis con Ventanas de Tiempo"** utiliza una representación de las ciudades como nodos en un mapa, donde los costos de viaje entre las ciudades están dados en forma de una matriz de adyacencia que especifica los tiempos de viaje entre pares de ciudades en horas. El objetivo principal es encontrar la ruta más corta posible cumpliendo con las restricciones de las ventanas de tiempo.

# Descripción del algoritmo genético híbrido

Un **algoritmo genético híbrido** es una variación de los algoritmos genéticos (AG) estándar que combina la exploración global característica de los AG con métodos adicionales, típicamente locales o heurísticos, para mejorar la eficiencia y la calidad de las soluciones encontradas. Este enfoque es ampliamente utilizado en problemas complejos como la optimización combinatoria, donde las técnicas estándar pueden no ser suficientemente rápidas o precisas para encontrar soluciones óptimas o cercanas al óptimo.

**Características clave:**

* **Exploración global**: Los AG generan y evolucionan soluciones usando selección, cruzamiento y mutación.
* **Optimización local**: Se aplican métodos de refinamiento, como búsqueda local o metaheurísticas (e.g., recocido simulado), para mejorar las soluciones.
* **Equilibrio exploración-explotación**: Integra la búsqueda amplia de los AG con el enfoque intensivo en áreas prometedoras de las heurísticas locales.

**Ventajas:**

* Encuentra soluciones de mejor calidad en menos tiempo.
* Acelera la convergencia hacia soluciones óptimas.
* Es útil para problemas complejos y con restricciones.

**Aplicación típica:**

En problemas como el **TSP con ventanas de tiempo (TSP-TW)**, los AG híbridos permiten explorar rutas globalmente y ajustar las soluciones para respetar las restricciones (ventanas de tiempo), logrando un balance entre exploración y refinamiento local.

# Parámetros usados

% Parámetros del Algoritmo Genético

numGeneraciones = 1000;

tamPoblacion = 50;

probMutacion = 0.05;

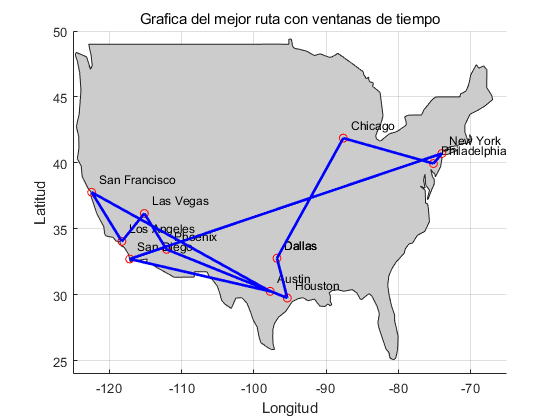
probCrossover = 0.9;

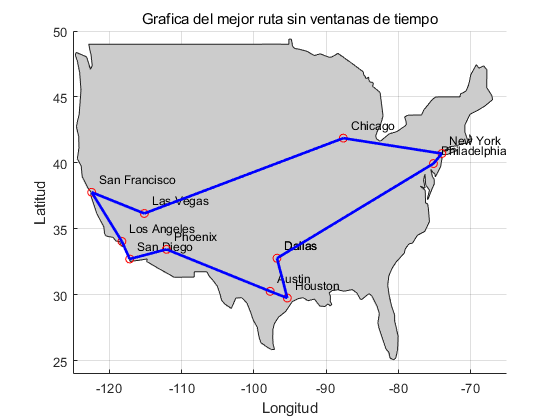
lambda = 50; % Factor de penalización

# Tabla con los resultados estadísticos

**Con ventana de tiempo:**

|  |  |  |  |
| --- | --- | --- | --- |
| Mejor ruta | Valor de la mejor ruta | Ruta promedio | Valor de la ruta promedio |
| 8 4 5 11 2 9 10 7 1 6 3 | 340.32 | 6 3 8 4 5 11 2 10 9 7 1 | 157260.32 |



**Sin ventana de tiempo:**

|  |  |  |  |
| --- | --- | --- | --- |
| Mejor ruta | Valor de la mejor ruta | Ruta promedio | Valor de la ruta promedio |
| 8 6 1 3 11 9 2 7 5 10 4 | 159.32 | 5 8 3 1 6 4 10 7 9 2 11 | 159.32 |

# Resultados obtenidos

El algoritmo identifica la ruta que minimiza la función de costo (distancia total y penalización por ventanas de tiempo).

Los resultados muestran una solución eficiente que respeta las restricciones de tiempo en la medida de lo posible.

**Función de Aptitud**:

La inclusión de penalizaciones por violación de ventanas de tiempo asegura que las restricciones sean consideradas, balanceando la exploración entre rutas cortas y factibles.

**Gráfica de la Mejor Ruta**:

La representación visual ayuda a validar que la solución es intuitivamente correcta, conectando ciudades en un orden lógico.

**Eficiencia**:

El algoritmo logra una convergencia efectiva mediante técnicas genéticas y asegura un balance razonable entre exploración y explotación, aunque podría beneficiarse de refinamientos adicionales como búsqueda local.

# Código

function tsp\_tw\_1()

% Parámetros del Algoritmo Genético

numGeneraciones = 1000;

tamPoblacion = 50;

probMutacion = 0.05;

probCrossover = 0.9;

lambda = 50; % Factor de penalización

% Datos del problema: Costos entre ciudades y ventanas de tiempo

distancias = [

0, 61.82, 18.54, 37.52, 54.08, 1.88, 59.98, 32.82, 69.42, 36.76, 60.26;

61.82, 0, 50.84, 33.62, 7.5, 59.88, 2.76, 28.84, 7.78, 28.14, 5.8;

18.54, 50.84, 0, 26.74, 43.38, 18.6, 49.28, 22, 58.7, 23.36, 49.3;

37.52, 33.62, 26.74, 0, 26.16, 35.56, 32.06, 4.8, 41.5, 3.26, 32.08;

54.08, 7.5, 43.38, 26.16, 0, 52.06, 7.32, 21.38, 15.34, 20.68, 5.92;

1.88, 59.88, 18.6, 35.56, 52.06, 0, 57.96, 30.86, 67.38, 34.8, 58.3;

59.98, 2.76, 49.28, 32.06, 7.32, 57.96, 0, 27.28, 10.62, 26.58, 6.76;

32.82, 28.84, 22, 4.8, 21.38, 30.86, 27.28, 0, 36.72, 4.02, 27.3;

69.42, 7.78, 58.7, 41.5, 15.34, 67.38, 10.62, 36.72, 0, 36.02, 12.14;

36.76, 28.14, 23.36, 3.26, 20.68, 34.8, 26.58, 4.02, 36.02, 0, 26.6;

60.26, 5.8, 49.3, 32.08, 5.92, 58.3, 6.76, 27.3, 12.14, 26.6, 0

];

% Ventanas de tiempo para cada ciudad (en horas)

ventanasTiempo = [

-inf, inf; % New York

50, 90; % Los Angeles

15, 25; % Chicago

30, 55; % Houston

15, 75; % Phoenix

5, 35; % Philadelphia

150, 200; % San Diego

25, 50; % Dallas

65, 100; % San Francisco

120, 150; % Austin

30, 85 % Las Vegas

];

% ventanasTiempo = [

% -inf, inf; % New York

% -inf, inf; % Los Angeles

% -inf, inf; % Chicago

% -inf, inf; % Houston

% -inf, inf; % Phoenix

% -inf, inf; % Philadelphia

% -inf, inf; % San Diego

% -inf, inf; % Dallas

% -inf, inf; % San Francisco

% -inf, inf; % Austin

% -inf, inf % Las Vegas

% ];

%Generar población inicial

poblacion = inicializarPoblacion(tamPoblacion, size(distancias, 1));

% Evolución del Algoritmo Genético

for generacion = 1:numGeneraciones

% Evaluar la aptitud de la población

aptitud = evaluarPoblacion(poblacion, distancias, ventanasTiempo, lambda);

% Selección

nuevaPoblacion = seleccion(poblacion, aptitud);

% Crossover

nuevaPoblacion = crossover(nuevaPoblacion, probCrossover);

% Mutación

nuevaPoblacion = mutacion(nuevaPoblacion, probMutacion);

% Reemplazar la población actual

poblacion = nuevaPoblacion;

end

% % Evaluar la mejor solución encontrada

aptitudFinal = evaluarPoblacion(poblacion, distancias, ventanasTiempo, lambda);

[~, mejorIndice] = min(aptitudFinal);

mejorRuta = poblacion(mejorIndice, :);

% % Mostrar la mejor ruta encontrada

% fprintf('Mejor ruta encontrada: ');

% disp(mejorRuta);

% fprintf('Costo total: %.2f\n', aptitudFinal(mejorIndice));

% Calcular la media de los valores de aptitud

promedioAptitud = mean(aptitudFinal);

% Encontrar el índice de la aptitud más cercana al promedio

[~, promIndice] = min(abs(aptitudFinal - promedioAptitud));

% Seleccionar la ruta promedio

promRuta = poblacion(promIndice, :);

% Mostrar los resultados

fprintf('Mejor ruta encontrada: ');

disp(mejorRuta);

fprintf('Costo total de la mejor ruta: %.2f\n', aptitudFinal(mejorIndice));

fprintf('\n\n');

fprintf('Ruta promedio encontrada: ');

disp(promRuta);

fprintf('Costo total de la ruta promedio: %.2f\n', aptitudFinal(promIndice));

grafica(mejorRuta)

end

% Funciones auxiliares

function poblacion = inicializarPoblacion(tamPoblacion, numCiudades)

% Inicializa la población aleatoriamente

poblacion = zeros(tamPoblacion, numCiudades);

for i = 1:tamPoblacion

poblacion(i, :) = randperm(numCiudades);

end

end

function aptitud = evaluarPoblacion(poblacion, distancias, ventanasTiempo, lambda)

% Evaluar la aptitud de cada ruta en la población considerando las ventanas de tiempo

tamPoblacion = size(poblacion, 1);

aptitud = zeros(tamPoblacion, 1);

for i = 1:tamPoblacion

ruta = poblacion(i, :);

% Encontrar el índice de la ciudad 1 en la ruta

idxInicio = find(ruta == 1, 1);

% Reorganizar la ruta para que empiece desde la ciudad 1

ruta = [ruta(idxInicio:end), ruta(1:idxInicio-1)];

ruta(end + 1) = 1; % Añadir la ciudad 1 al final para el regreso

% Inicialización de tiempos

tiempoLlegada = zeros(1, length(ruta));

penalizacion = 0;

% Evaluar la ruta completa

for j = 2:length(ruta)

ciudadAnterior = ruta(j - 1);

ciudadActual = ruta(j);

% Calcular el tiempo de viaje y llegada

tiempoViaje = distancias(ciudadAnterior, ciudadActual);

tiempoLlegada(j) = max(ventanasTiempo(ciudadActual, 1), tiempoLlegada(j - 1) + tiempoViaje);

% Penalización por exceder la ventana de tiempo superior

exceso = max(0, tiempoLlegada(j) - ventanasTiempo(ciudadActual, 2));

penalizacion = penalizacion + exceso^2; % Penalización cuadrática

end

% Calcular el tiempo total de la ruta y agregar la penalización

tiempoTotal = tiempoLlegada(end);

aptitud(i) = tiempoTotal + lambda \* penalizacion;

end

end

function nuevaPoblacion = seleccion(poblacion, aptitud)

% Selección por torneo

tamPoblacion = size(poblacion, 1);

nuevaPoblacion = poblacion;

for i = 1:tamPoblacion

idx1 = randi(tamPoblacion);

idx2 = randi(tamPoblacion);

if aptitud(idx1) < aptitud(idx2)

nuevaPoblacion(i, :) = poblacion(idx1, :);

else

nuevaPoblacion(i, :) = poblacion(idx2, :);

end

end

end

function nuevaPoblacion = crossover(poblacion, probCrossover)

% Operador de crossover cíclico para toda la población

tamPoblacion = size(poblacion, 1);

nuevaPoblacion = poblacion;

for i = 1:2:tamPoblacion-1

if rand < probCrossover

% Seleccionar dos padres y realizar el cruzamiento cíclico

padre1 = poblacion(i, :);

padre2 = poblacion(i+1, :);

[hijo1, hijo2] = cruzamientoCiclico(padre1, padre2);

nuevaPoblacion(i, :) = hijo1;

nuevaPoblacion(i+1, :) = hijo2;

end

end

end

function [descendiente1, descendiente2] = cruzamientoCiclico(padre1, padre2)

% Esta función implementa el cruzamiento cíclico (Cycle Crossover, CX)

% Entrada:

% padre1 - vector del primer padre

% padre2 - vector del segundo padre

% Salida:

% descendiente1 - vector del primer descendiente

% descendiente2 - vector del segundo descendiente

% Inicialización de descendientes

descendiente1 = zeros(1, length(padre1));

descendiente2 = zeros(1, length(padre2));

% Identificación de ciclos

ciclo\_inicial = 1; % Comienza desde el primer índice no visitado

visitados = false(1, length(padre1)); % Marcador para posiciones visitadas

while any(~visitados)

% Ciclo actual

ciclo\_indices = [];

indice\_actual = find(~visitados, 1); % Encuentra el primer índice no visitado

inicio = indice\_actual; % Guarda el inicio del ciclo

% Construcción del ciclo

while true

ciclo\_indices(end + 1) = indice\_actual;

visitados(indice\_actual) = true;

valor = padre2(indice\_actual);

indice\_actual = find(padre1 == valor);

if indice\_actual == inicio

break;

end

end

% Asignación a los descendientes según el ciclo

if mod(ciclo\_inicial, 2) == 1 % Ciclos impares: copia de Padre 1 a Descendiente 1

descendiente1(ciclo\_indices) = padre1(ciclo\_indices);

descendiente2(ciclo\_indices) = padre2(ciclo\_indices);

else % Ciclos pares: intercambio de padres

descendiente1(ciclo\_indices) = padre2(ciclo\_indices);

descendiente2(ciclo\_indices) = padre1(ciclo\_indices);

end

% Avanzar al siguiente ciclo

ciclo\_inicial = ciclo\_inicial + 1;

end

% Rellenar las posiciones restantes

descendiente1(descendiente1 == 0) = padre2(descendiente1 == 0);

descendiente2(descendiente2 == 0) = padre1(descendiente2 == 0);

end

function nuevaPoblacion = mutacion(poblacion, probMutacion)

% Operador de mutación (intercambio de dos ciudades)

tamPoblacion = size(poblacion, 1);

numCiudades = size(poblacion, 2);

nuevaPoblacion = poblacion;

for i = 1:tamPoblacion

if rand < probMutacion

idx1 = randi(numCiudades);

idx2 = randi(numCiudades);

% Intercambiar dos ciudades en la ruta

temp = nuevaPoblacion(i, idx1);

nuevaPoblacion(i, idx1) = nuevaPoblacion(i, idx2);

nuevaPoblacion(i, idx2) = temp;

end

end

end

%%%%%%%%%%%%%%%%%%%%%% GRAFICA %%%%%%%%%%%%%%%%%%%

function grafica(ciudades\_conectadas)

% Coordenadas de las ciudades (lat, lon)

ciudades = {

'New York', 40.7128, -74.0060;

'Los Angeles', 34.0522, -118.2437;

'Chicago', 41.8781, -87.6298;

'Houston', 29.7604, -95.3698;

'Phoenix', 33.4484, -112.0740;

'Philadelphia', 39.9526, -75.1652;

'San Diego', 32.7157, -117.1611;

'Dallas', 32.7767, -96.7970;

'San Francisco', 37.7749, -122.4194;

'Austin', 30.2672, -97.7431;

'Las Vegas', 36.1699, -115.1398

};

% Arreglo de ciudades que final regrese al inicio

ciudades\_conectadas = [ciudades\_conectadas, ciudades\_conectadas(1)];

% Cargar el archivo shapefile de los países

shapefile\_path = '110m\_cultural/ne\_110m\_admin\_0\_countries.shp'; % Reemplaza con la ruta correcta

S = shaperead(shapefile\_path);

% Filtrar solo los datos de Estados Unidos

usa = S(strcmp({S.NAME}, 'United States of America'));

% Crear la figura

figure;

hold on;

% Mostrar el mapa de los EE.UU. de fondo

geoshow(usa, 'FaceColor', [0.8 0.8 0.8]);

% Graficar las ciudades con sus coordenadas

for i = 1:length(ciudades\_conectadas)

ciudad\_id = ciudades\_conectadas(i);

nombre = ciudades{ciudad\_id, 1};

lat = ciudades{ciudad\_id, 2};

lon = ciudades{ciudad\_id, 3};

plot(lon, lat, 'ro'); % 'ro' para marcar las ciudades con puntos rojos

text(lon + 1, lat + 1, nombre, 'FontSize', 9); % Etiquetar las ciudades

end

% Conectar las ciudades en el orden especificado

for i = 1:length(ciudades\_conectadas) - 1

ciudad\_inicio = ciudades\_conectadas(i);

ciudad\_fin = ciudades\_conectadas(i + 1);

lat1 = ciudades{ciudad\_inicio, 2};

lon1 = ciudades{ciudad\_inicio, 3};

lat2 = ciudades{ciudad\_fin, 2};

lon2 = ciudades{ciudad\_fin, 3};

plot([lon1 lon2], [lat1 lat2], 'b-', 'LineWidth', 2); % Línea azul conectando las ciudades

end

% Ajustes del gráfico

title('Grafica del mejor ruta');

xlabel('Longitud');

ylabel('Latitud');

xlim([-125, -65]); % Limites de longitud para EE.UU.

ylim([24, 50]); % Limites de latitud para EE.UU.

grid on;

hold off;

end