C3 OPTIMIZACIÓN DE RUTAS ROBÓTICAS EN MANUFACTURA DE ALTA PRECISIÓN

December 1, 2025

1 Contexto

En una empresa que fabrica placas de circuito impreso (PCB), un brazo robótico debe soldar componentes en varios puntos predefinidos en la placa. Para maximizar la producción, se necesita minimizar el tiempo total del proceso.

El tiempo está dominado por el movimiento del brazo, por lo que el objetivo es encontrar la **ruta más corta** que visite todos los puntos de soldadura una sola vez, comenzando y terminando en un punto base.

El espacio de búsqueda consiste en todas las permutaciones posibles de los puntos de soldadura. Para N puntos, hay $(N-1)!/2$ rutas posibles, un número que crece de forma exponencial, haciendo inviable una búsqueda exhaustiva. Este es un problema ideal para ser abordado con el algoritmo **simulated annealing**.

Objetivos:

1. Definir la función objetivo, la función debe calcular la longitud total de una ruta.
2. Implementar **simulated annealing** para encontrar una ruta de menor costo, es decir, la ruta más corta.

2 Fundamento matemático

El problema se modela matemáticamente como una búsqueda de permutación óptima en un grafo completo ponderado.

Sea $P = \{p_1, p_2, \dots, p_N\}$ el conjunto de puntos de soldadura, donde cada punto p_i tiene coordenadas (x_i, y_i) en el plano 2D del PCB.

La distancia entre dos puntos cualesquiera i y j se define como la distancia Euclidiana:

$$d(i,j) = \|p_i - p_j\|_2 = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Buscamos una permutación σ de los índices $\{1, 2, \dots, N\}$ que represente el orden de la visita. La función objetivo $L(\sigma)$ es la suma de las distancias del recorrido, incluyendo el retorno al punto de inicio (ciclo cerrado):

$$L(\sigma) = \sum_{i=1}^{N-1} d(\sigma(k), \sigma(k+1)) + d(\sigma(N), \sigma(1))$$

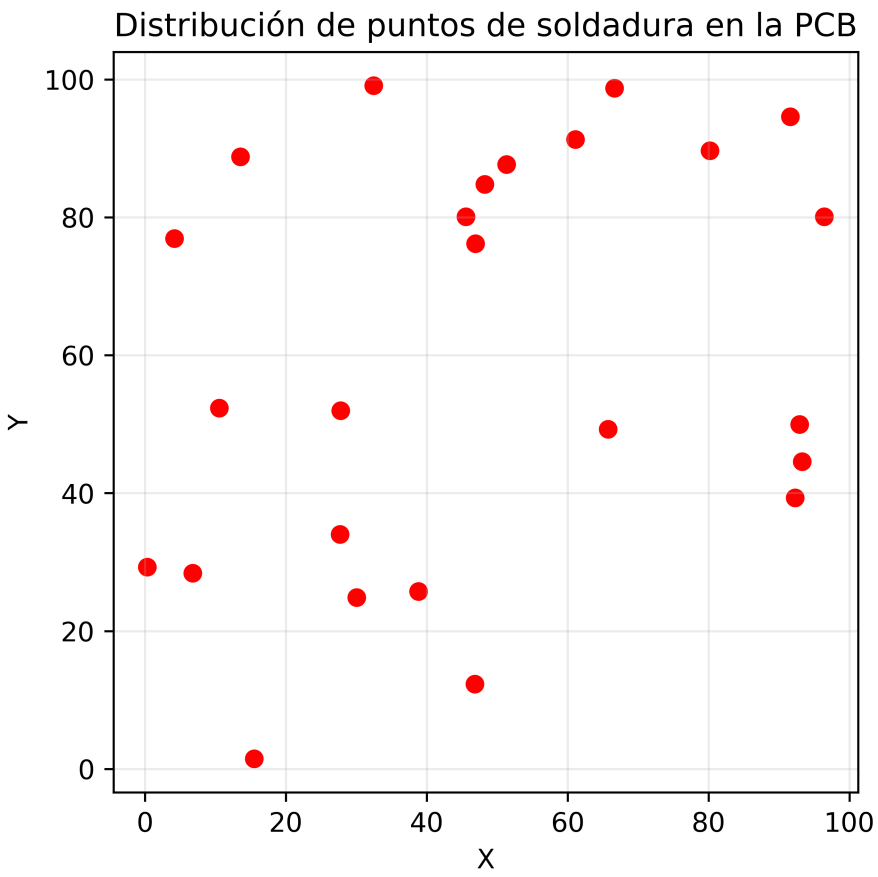
3 Implementación

[1]:
```
# Cargar librerías necesarias
# ---------------------------
import numpy as np
import matplotlib.pyplot as plt

# Configuraciones iniciales
# -------------------------
plt.rcParams["figure.dpi"] = 600
```

[2]:
```
# Conjunto de datos (simulados)
# -----------------------------
np.random.seed(2025)

# Parámetros del problema
n_puntos = 25

# Generación de coordenadas
puntos = np.random.uniform(low = 0, high = 100, size = (n_puntos, 2))
print("Primeros 5 puntos de soldadura:\n", puntos[:5])
```

```
Primeros 5 puntos de soldadura:
 [[13.54881637 88.78517027]
 [93.26056399 44.5568164 ]
 [38.82355461 25.75964353]
 [65.73675855 49.26169375]
 [96.42384193 80.09844749]]
```

[3]:
```
# Visualización inicial de los puntos
fig, ax = plt.subplots(figsize = (5, 5))

ax.scatter(puntos[:, 0], puntos[:, 1], color = "red", marker = "o")
ax.set_title("Distribución de puntos de soldadura en la PCB")
ax.set_xlabel("X")
ax.set_ylabel("Y")
plt.grid(True, alpha = 0.25)
plt.show()
```



[4]:
```python
# Definición del función objetivo
# ------------------------------
def distancia_total(ruta, coordenadas):
    distancia = 0.0
    n_puntos = len(ruta)
    for i in range(n_puntos):
        id_actual = ruta[i]
        id_siguiente = ruta[(i + 1) % n_puntos]
        p1 = coordenadas[id_actual]
        p2 = coordenadas[id_siguiente]
        d = np.sqrt(np.sum((p1 - p2) ** 2))
        distancia = distancia + d
    return distancia
```

[5]:
```python
# Ejemplo: calcular distancia de una ruta aleatoria
ruta_inicial = np.arange(n_puntos)
np.random.shuffle(ruta_inicial)
distancia_inicial = distancia_total(ruta_inicial, puntos)
print("Distancia (aleatoria):", distancia_inicial)
```

```
Distancia (aleatoria): 1267.8058707750697
```

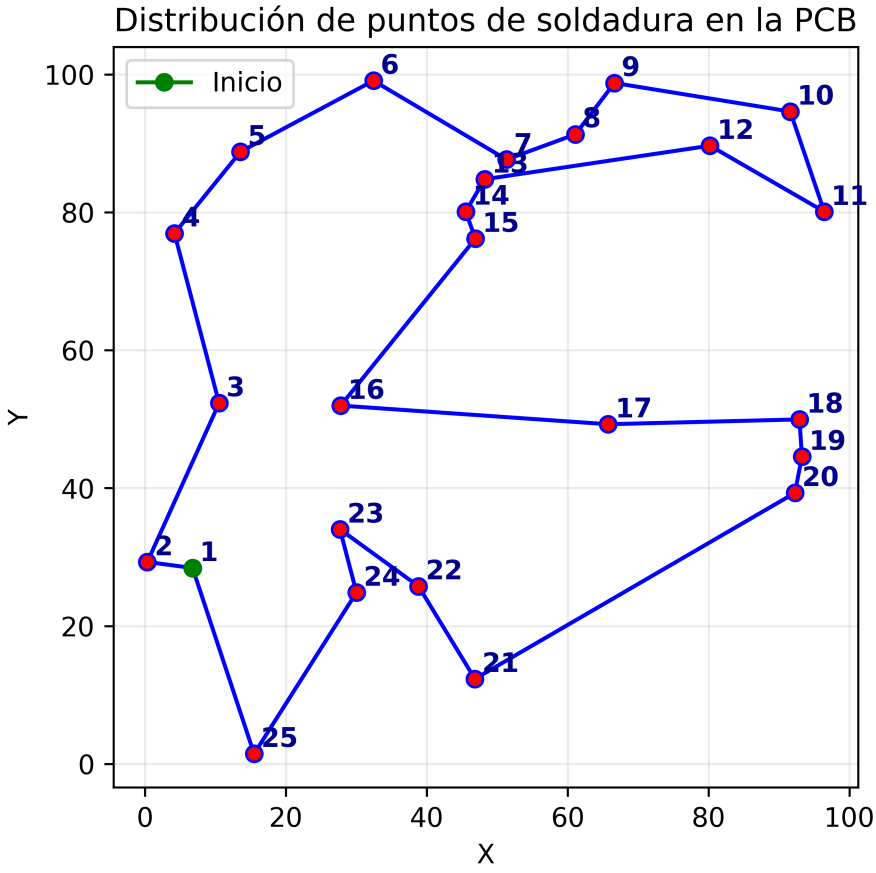
[6]:
```python
def simulated_annealing(coordenadas, t_inicial = 10000, t_final = 1e-6, alpha =␣
↪0.999):
    n = len(coordenadas)
    r_actual = np.arange(n)
    np.random.shuffle(r_actual)
    f_actual = distancia_total(r_actual, coordenadas)
    r_mejor = r_actual.copy()
    f_mejor = f_actual
    temperatura = t_inicial
    iteraciones = 0
    while temperatura > t_final:
        r_nuevo = r_actual.copy()
        i, j = np.random.choice(r_nuevo, size = 2, replace = False)
        r_nuevo[i], r_nuevo[j] = r_nuevo[j], r_nuevo[i]
        f_nuevo = distancia_total(r_nuevo, coordenadas)
        delta = f_nuevo - f_actual
        if delta < 0:
            r_actual = r_nuevo
            f_actual = f_nuevo
            if f_actual < f_mejor:
                r_mejor = r_actual.copy()
                f_mejor = f_actual
        else:
            probabilidad = np.exp(-delta / temperatura)
            if np.random.rand() < probabilidad:
                r_actual = r_nuevo
                f_actual = f_nuevo
        temperatura = alpha * temperatura
        iteraciones = iteraciones + 1
    return r_mejor, f_mejor, iteraciones
```

[7]:
```python
# Ejecución del algoritmo
# -----------------------
mejor_ruta, mejor_distancia, iter = simulated_annealing(puntos)
print("Mejor distancia encontrada:", mejor_distancia)
print("Iteraciones:", iter)
```

```
Mejor distancia encontrada: 491.4230591978067
Iteraciones: 23015
```

[8]:
```python
x_ordenado = puntos[mejor_ruta, 0]
y_ordenado = puntos[mejor_ruta, 1]

x_plot = np.append(x_ordenado, x_ordenado[0])
y_plot = np.append(y_ordenado, y_ordenado[0])
```

```
fig, ax = plt.subplots(figsize = (5, 5))
ax.plot(x_plot, y_plot, color = "blue", marker = "o", mfc = "red")
ax.plot(x_ordenado[0], y_ordenado[0], color = "green", marker = "o", label =␣
 ↪"Inicio")

for i in range(len(mejor_ruta)):
    ax.text(x_ordenado[i] + 1, y_ordenado[i] + 1, i + 1,
            color = "darkblue", fontweight = "bold")

ax.set_title("Distribución de puntos de soldadura en la PCB")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
plt.grid(True, alpha = 0.25)
plt.show()
```



Juan F. Olivares Pacheco (jfolivar@uda.cl)