01

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

EDA: Estimation Distribution Algorithms

Resolviendo el Problema del Agente Viajero

Juan Ramirez 20666
Jonathan Espinoza 20022

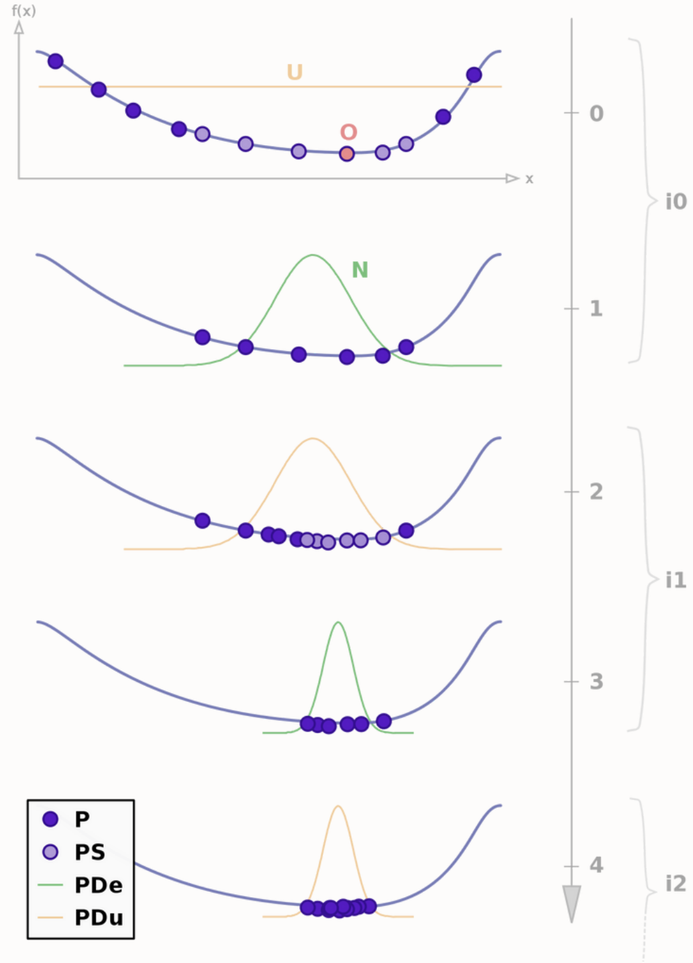
METODOS NÚMERICOS 2
VERSIÓN 1.0 - 2023

Índice

EDA

- Métodos de optimización estocástica que guían la búsqueda del óptimo mediante la construcción y muestreo de modelos probabilísticos. Estos modelos de probabilidad explícitos, buscan regiones de soluciones candidatas prometedoras.

- **Generación de Muestra:** En cada iteración, se crea una muestra de soluciones utilizando una distribución de probabilidad específica.
- **Evaluación y Selección:** Las soluciones en la muestra se evalúan según una función objetivo, y se seleccionan las mejores.
- **Actualización de la Distribución:** La distribución de probabilidad se ajusta basándose en las soluciones seleccionadas.
- **Repetición del Proceso:** Se repite el proceso, generando nuevas muestras con la distribución actualizada, hasta cumplir un criterio de finalización.



UMDA

Univariate Marginal Distribution Algorithm

Las EDA más simples asumen que las variables de decisión son independientes, es decir, p(X1 , X2) = p(X1) p(X2). Por lo tanto, las EDA univariadas se basan solo en estadísticas univariadas y las distribuciones multivariadas deben factorizarse como producto de n distribuciones univariadas.

$$D_{\text{Univariate}} = p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i).$$

Operador alpha UMDA:

$$p_{t+1}(X_i) = \frac{1}{k} \sum_{x \in S(P(t))} x_i$$

Implementación

UMDA

- **Inicialización de la Población Inicial:** La población inicial se genera de manera aleatoria, donde cada individuo representa una posible solución al TSP.
- **Cálculo de la Matriz de Distancias:** Calculamos la matriz de distancias entre todas las ciudades para evaluar las rutas.
- **Selección de las Mejores Rutas:** Las rutas se evalúan por su distancia total, seleccionando las m´as cortas para la próxima generación
- **Estimación de la Distribución de Probabilidades:** La distribución de probabilidad se actualiza según las rutas seleccionadas.
- **Muestreo de Rutas Basado en Probabilidades:** Generamos nuevas rutas muestreando según la distribución de probabilidad estimada.

```
def ejecutar(porcentaje_aleatorio):
    # Para cada generación hasta num_generaciones
        # Evaluar y seleccionar mejores rutas
        # Estimar nueva distribución de probabilidades
        # Generar nueva población muestreando de esta distribución
        # Añadir rutas aleatorias para diversidad
        # Actualizar población con nuevas rutas
```

cGA

Compact Genetic Algorithm

El CGA es un algoritmo que se basa en poblaciones implícitas definidas por distribuciones univariadas. En este contexto, el algoritmo funciona de la siguiente manera:

1. Poblaciones Implícitas y Distribuciones Univariadas
   - cGA usa un modelo probabilístico en lugar de una población explícita de soluciones.
   - Cada característica se considera independientemente.
2. Estimación de Probabilidades
   - Actualiza probabilidades basadas en la comparación de las mejores y peores soluciones.
   - Incrementa la probabilidad de características de la mejor solución, disminuye la de la peor.
3. Aprendizaje y Actualización
   - La tasa de aprendizaje controla la rapidez de la actualización del modelo probabilístico.
   - Importante para equilibrar la velocidad de convergencia y la calidad de las soluciones.

$$p_{t+1}(X_i) = p_t(X_i) + \gamma(u_i - v_i), \quad \forall i \in 1, 2, \ldots, N,$$

Implementación

cGA

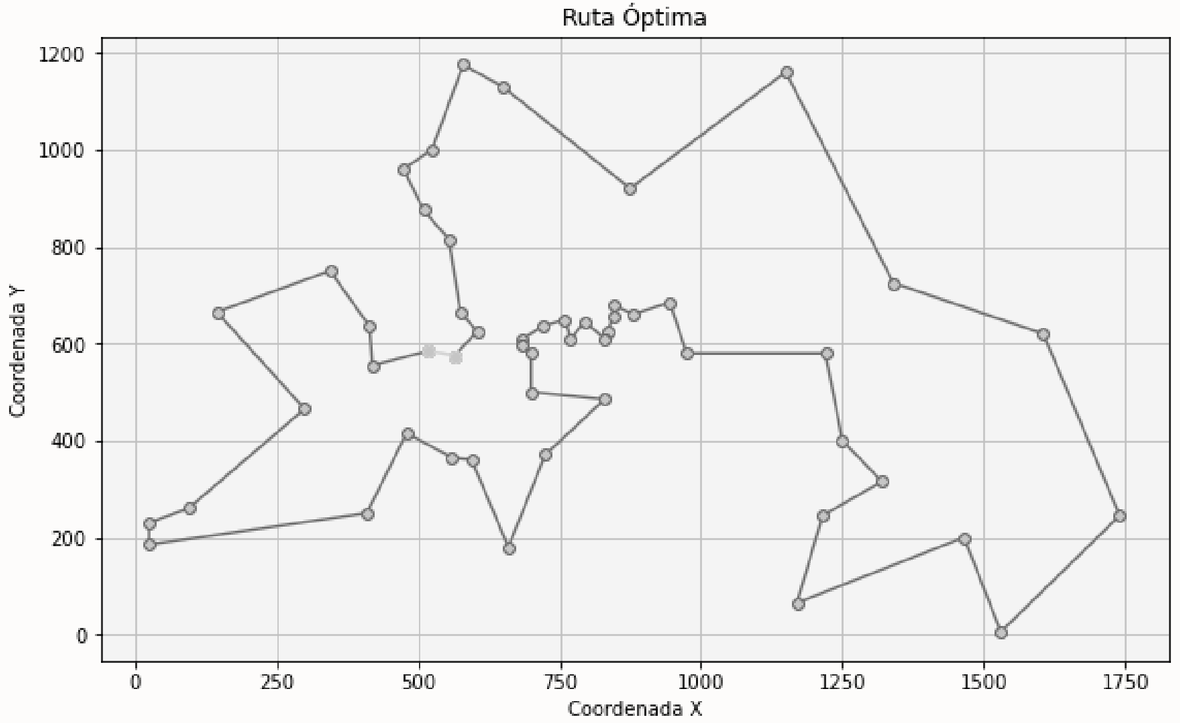
- **Inicialización del Vector Probabilístico:** Se inicializa un vector que representa la probabilidad de cada característica (por ejemplo, una posición en una ruta) de ser seleccionada.
- **Generación de Individuos:** A partir del vector probabilístico, se generan individuos (soluciones) de manera estocástica, eligiendo características basadas en las probabilidades.
- **Evaluación de Individuos:** Cada individuo generado se evalúa según una función objetivo, que en el caso del TSP sería la longitud total de la ruta.
- **Selección y Torneo:** Se seleccionan individuos para un torneo, usualmente los mejores y los peores, basándose en sus evaluaciones.
- **Actualización del Vector Probabilístico:** El vector se actualiza para aumentar la probabilidad de las características presentes en los mejores individuos y disminuir la de las presentes en los peores.
- **Iteración y Convergencia:** Estos pasos se repiten por un número predefinido de generaciones o hasta que el vector probabilístico converge, indicando que se ha encontrado una solución óptima o cercana al óptimo.

```
def ejecutar():
    #para cada generación hasta num_generaciones hacer:
    #Crear un torneo con individuos generados aleatoriamente
    #Evaluar las distancias de cada individuo en el torneo
    #Determinar los índices del ganador y del perdedor en el torneo
    #Seleccionar el ganador y el perdedor del torneo
    #Actualizar el vector probabilístico
   #Registrar la mejor ruta y su distancia agregar ganador a
mejores_rutas_por_generacion agregar distancia del ganador a
distancias_mejores_rutas
  #Determinar la mejor ruta y su distancia de todas las
  #Devolver la mejor ruta y su distancia devolver mejor_ruta,
mejor_distancia
```

Problema del Agente Viajero

Problema de optimización para encontrar la ruta más corta que pasa por un conjunto de puntos una sola vez y regresa al inicio.

- **Complejidad NP-hard:** Aumento factorial del número de rutas posibles con cada ciudad añadida, lo que lo hace computacionalmente desafiante.
- **Métodos Exactos:** Programación lineal, búsqueda exhaustiva.
- **Métodos Aproximados:** Algoritmos genéticos, búsqueda local, recocido simulado, EDA.



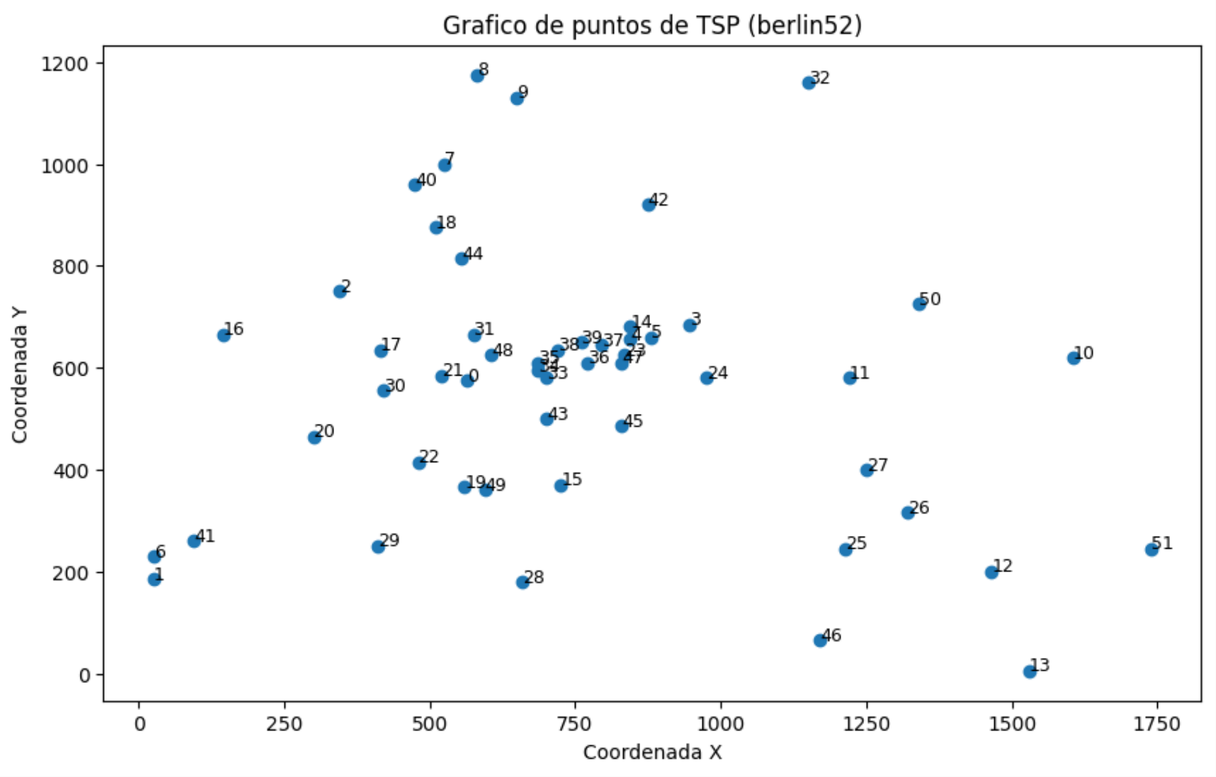
Problema del Agente Viajero

BERLIN 52

Origen de Datos: Las 52 ubicaciones corresponden a puntos de entrega reales en Berlín. Este conjunto de datos se ha convertido en un caso de prueba estándar para algoritmos de TSP.

Complejidad: Con 52 ciudades, hay un número enormemente grande de posibles rutas (10^60), pero aún es manejable para algunos algoritmos de optimización. Por lo tanto, "Berlin52" se utiliza comúnmente para probar y comparar la eficacia y eficiencia de varios algoritmos de solución de TSP.

Solución Óptima Conocida: A diferencia de muchas instancias de TSP donde la solución óptima no se conoce o es difícil de calcular, para "Berlin52" se ha encontrado una solución óptima.



OBJETIVOS

- Implementar el algoritmo Univariate Marginal Distribution Algorithm (UMDA) para resolver el Problema del Agente Viajero (TSP) en el conjunto de datos Berlin52 y evaluar su eficacia.
- Implementar el algoritmo CGA para resolver el Problema del Agente Viajero (TSP) en el conjunto de datos Berlin52 y evaluar su desempeño.
- Comparar el desempeño de ambos algoritmos tanto en tiempo como en resultados de los algoritmos univariantes de estimación de distribución entre sí.

Algoritmos

UMDA

Parametros:

- Población: 5000
- Generaciones: 100
- Tasa Selección: 10%

1. Inicialización y Generación de la Población Inicial
2. Cálculo de la Matriz de Distancias
3. Selección de las Mejores Rutas
4. Estimación de la Distribución de Probabilidades
5. Muestreo de Rutas Basado en Probabilidades
6. Ejecución del Algoritmo

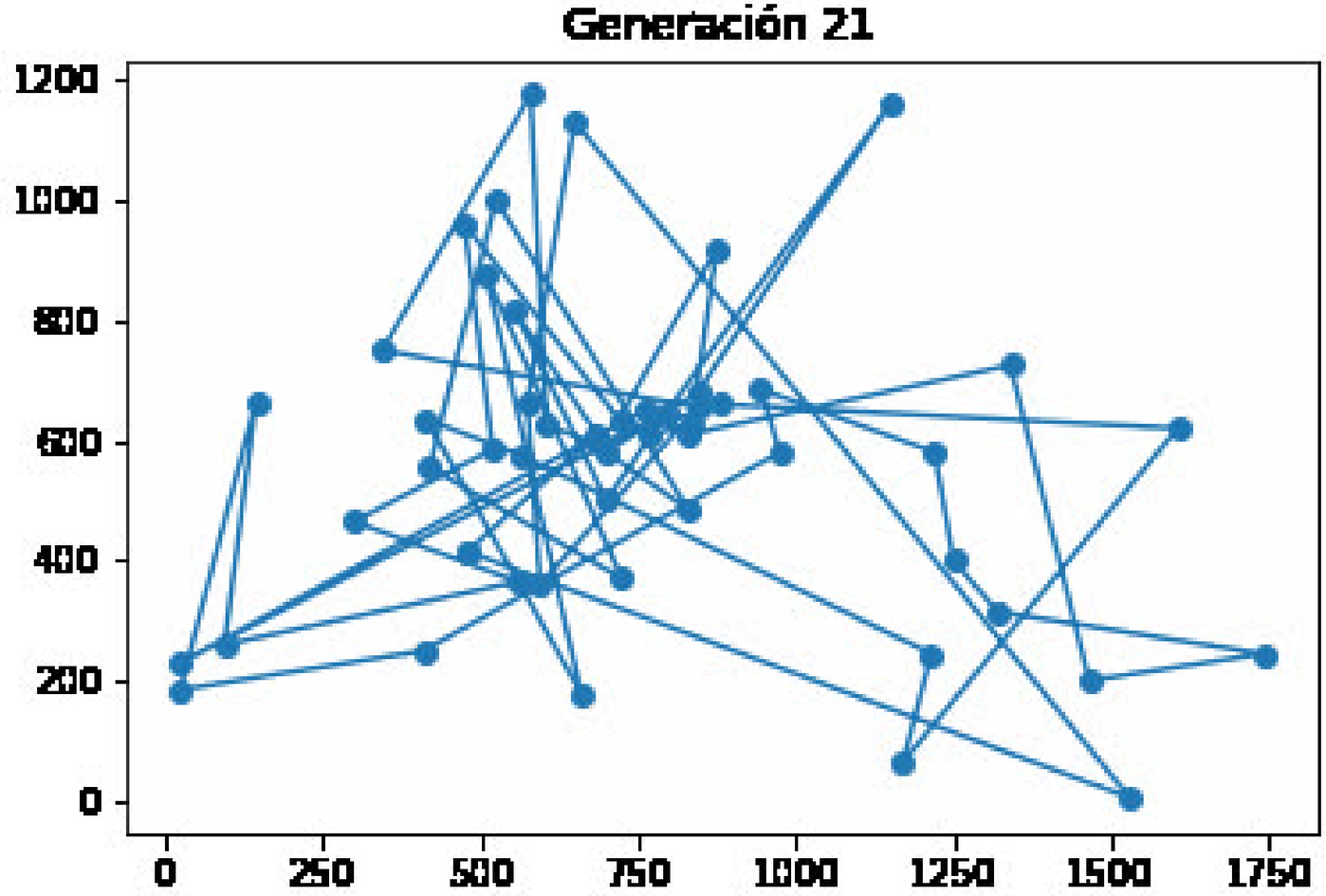
cGA

Parametros:

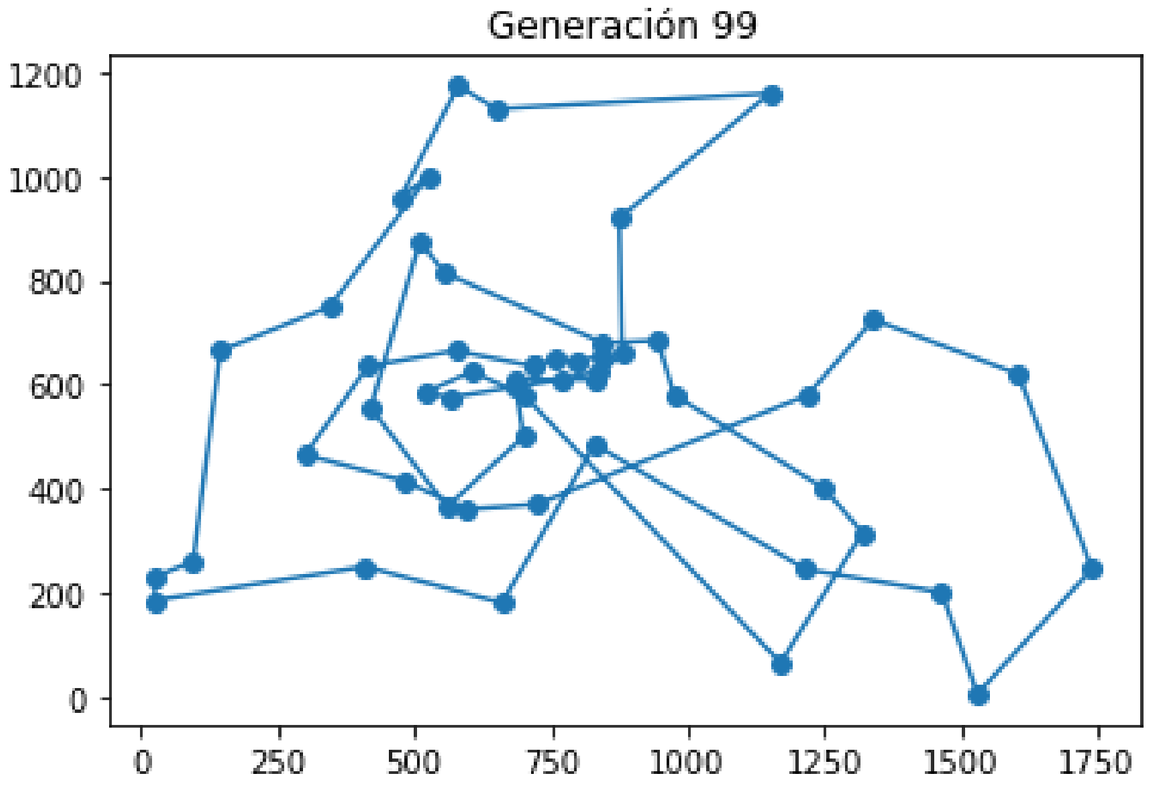
- Población: 500
- Generaciones: 1000
- Tasa Selección: 10%

1. Inicialización del Vector Probabilístico
2. Generación de Individuos
3. Evaluación de Individuos
4. Selección y Torneo
5. Actualización del Vector Probabilístico
6. Iteración y Convergencia

Resultados UMDA

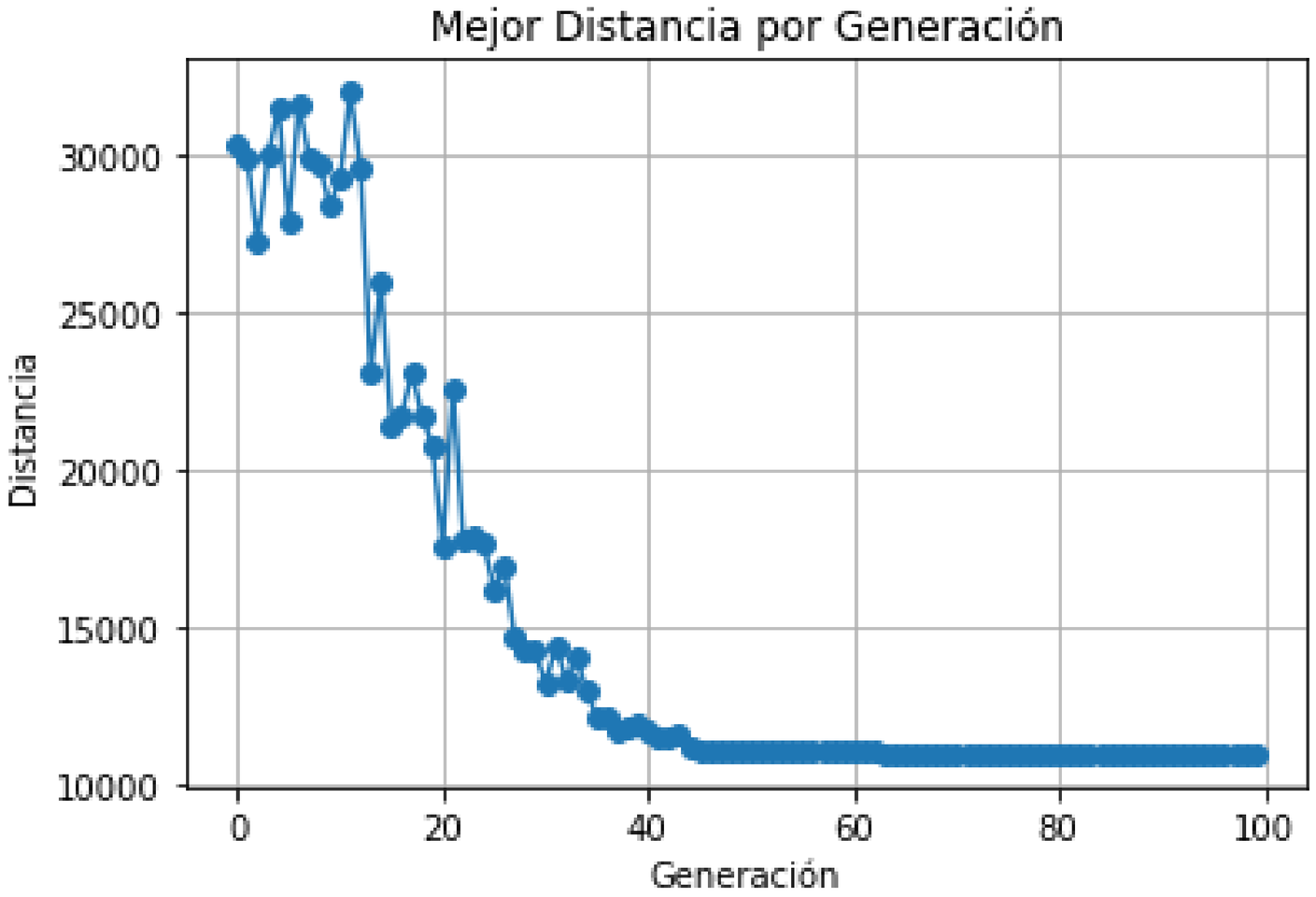


Resultados UMDA

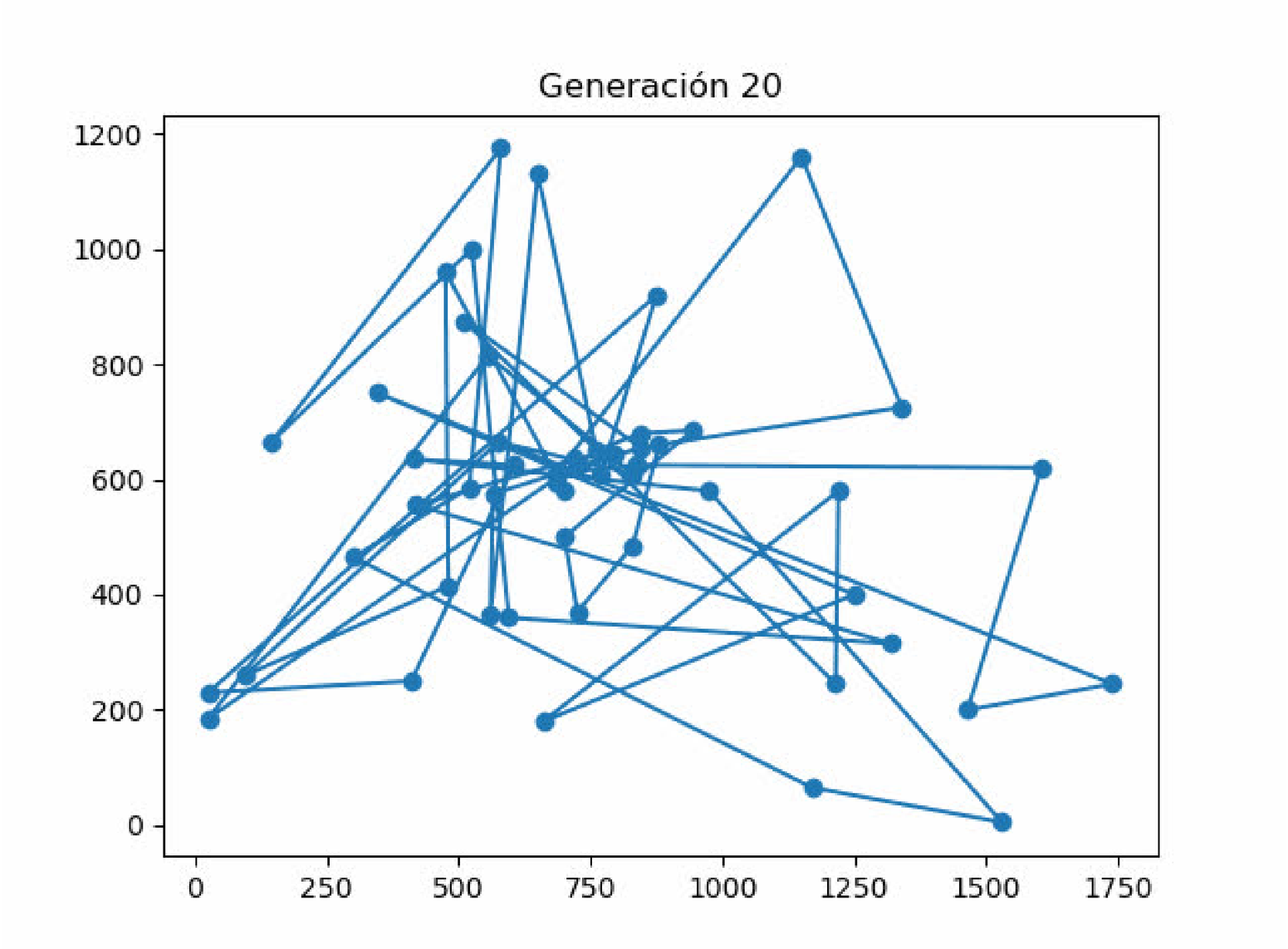


Distancia: 10970.85 unidades
Tiempo de ejecución: 14:51s

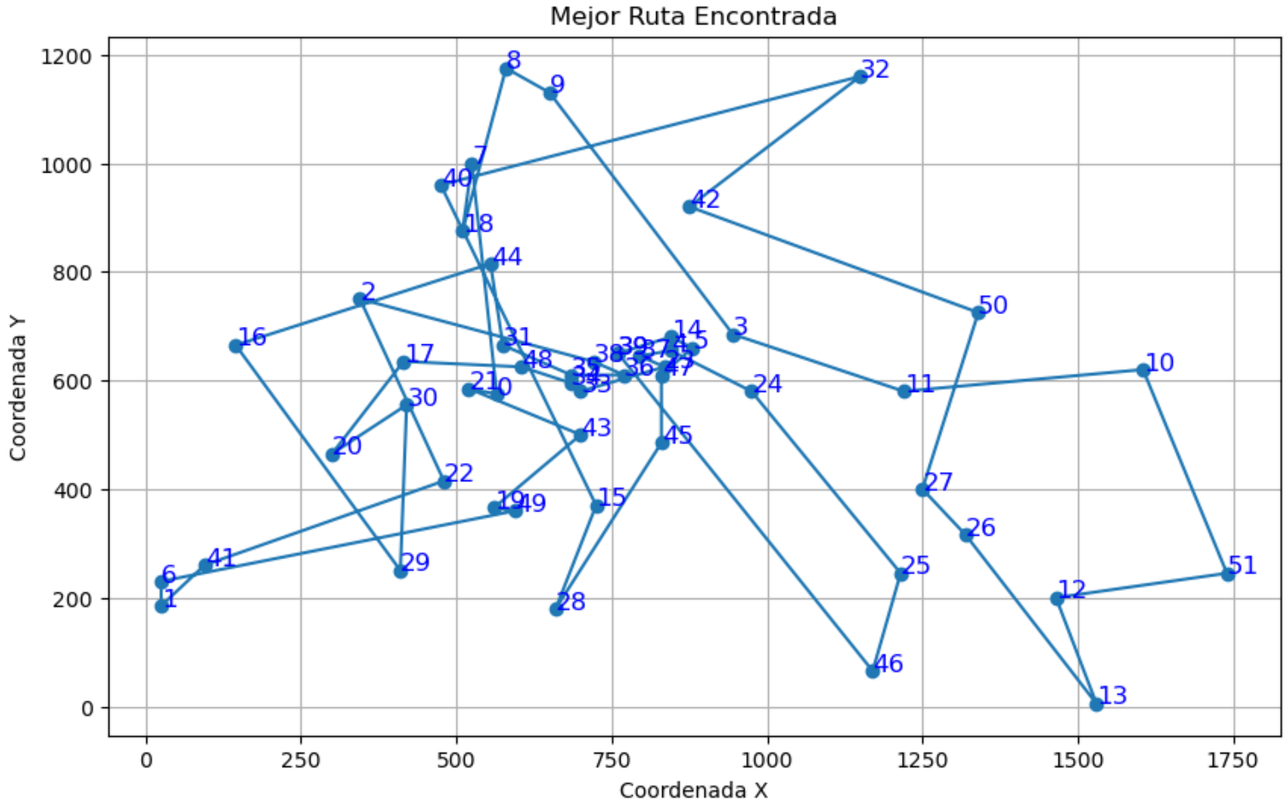
Resultados UMDA



Resultados cGA

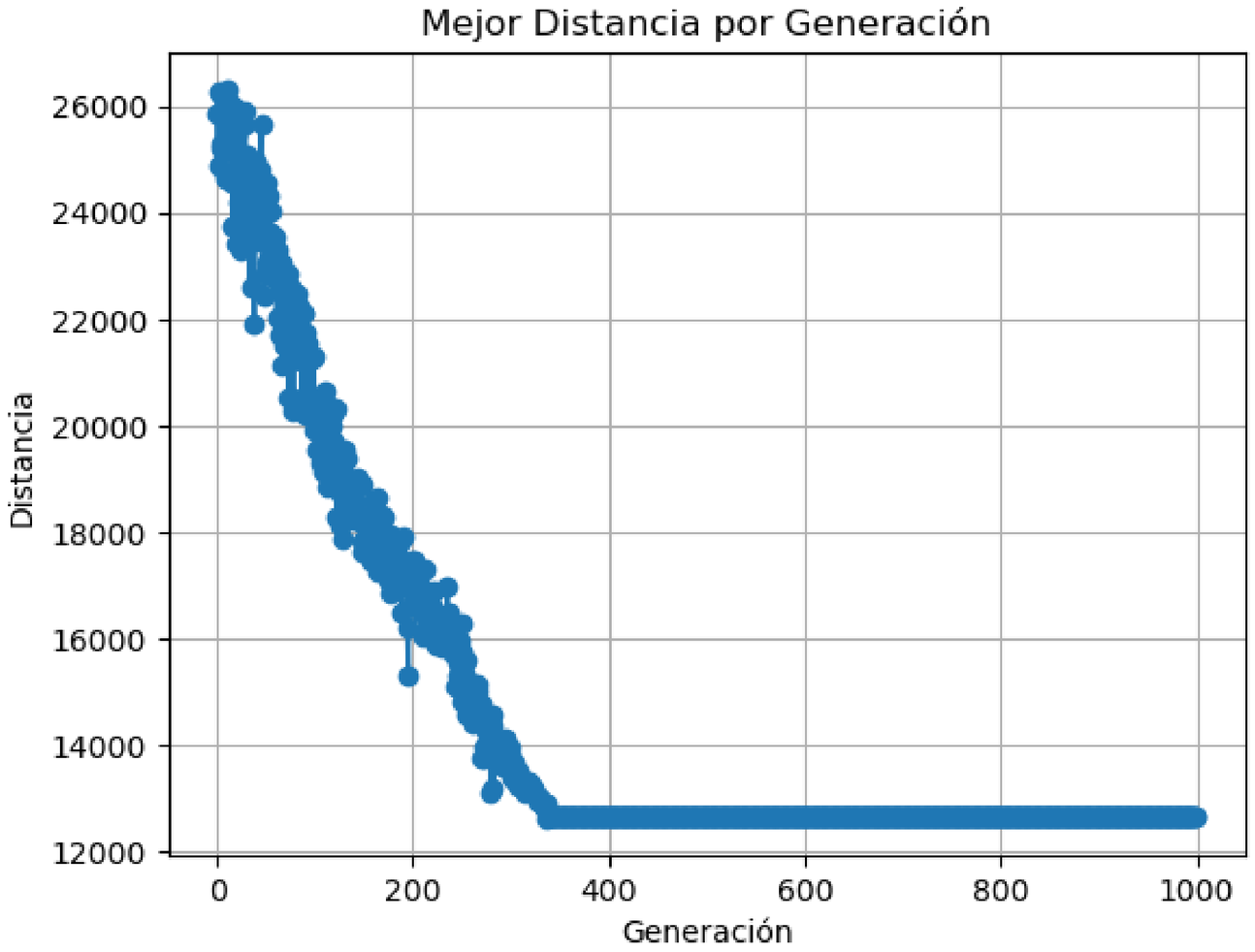


Resultados cGA



Distancia: 12617.0143 unidades
Tiempo de ejecución: 13:04 s

Resultados cGA



Discusión

- UMDA supera significativamente a CGA en términos de calidad de la solución para el Problema del Agente Viajero en este conjunto de datos.

- UMDA logró una solución de alta calidad en un tiempo de ejecución más corto en comparación con CGA, lo que indica una mayor eficiencia computacional.

Conclusiones

- UMDA es más efectivo en la búsqueda de soluciones óptimas para el TSP en Berlin52 en comparación con CGA.
- Además de la calidad de la solución, UMDA también demuestra ser más eficiente en términos de tiempo de ejecución.
- La elección entre CGA y UMDA debe basarse en las necesidades y objetivos específicos del problema, considerando tanto la calidad de la solución como la eficiencia computacional.

Recomendaciones

- .Explorar más variaciones de Algoritmos y Parámetros.
- Considerar la Escalabilidad y los Recursos Computacionales.
- Explorar Diversas Variantes del TSP.

Referencias

- Harik, G. R., Lobo, F. G., & Goldberg, D. E. (1999). The Compact Genetic Algorithm. IEEE Transactions on Evolutionary Computation, 3(4), [287]-[297].
- Reyes-Figueroa, A. (2021, 17 de noviembre). EDA: Estimation Distribution Algorithms [Presentación en clase de Métodos Numéricos II (Aula 33)].