Universidad Nacional del Centro de la

Provincia de Buenos Aires

FACULTAD DE CIENCIAS EXACTAS

Ingeniería de Sistemas

Introducción a la Computación Evolutiva

Francisco Tomas Mauri

2025

Indice

1. Introducción

El Problema del Viajante (TSP, por sus siglas en inglés) es uno de los problemas clásicos de optimización combinatoria, en el cual se busca determinar la ruta de menor costo que permita a un viajante visitar un conjunto de ciudades exactamente una vez y regresar a la ciudad de origen. Debido a la complejidad y la naturaleza NP-hard del TSP, se han propuesto diversas técnicas heurísticas y metaheurísticas para abordar su resolución.

En este proyecto se implementa un algoritmo evolutivo orientado a resolver instancias asimétricas del TSP, a partir de archivos ATSP. El uso de algoritmos evolutivos permite explorar de manera eficiente el espacio de soluciones mediante operadores inspirados en la genética (selección, cruce y mutación), combinados con estrategias de selección de sobrevivientes y condiciones de corte que garantizan la convergencia hacia soluciones de alta calidad.

El objetivo principal del proyecto es desarrollar, implementar y evaluar computacionalmente un algoritmo evolutivo, analizando la efectividad y eficiencia de distintas configuraciones de parámetros y componentes. Además, se incorpora un mecanismo de registro de resultados que documenta la configuración utilizada, la evolución del fitness a lo largo del tiempo, la mejor solución obtenida y el tiempo total de ejecución.

2. Desarrollo

2.1 Descripción del Problema

El Problema del Viajante se define de la siguiente manera: dado un conjunto de N ciudades y una matriz de costos C[i][j] que representa el costo de viajar de la ciudad i a la ciudad j, se busca determinar una permutación de las ciudades que minimice el costo total del recorrido, incluyendo el retorno a la ciudad de inicio. En nuestro caso, el archivo ATSP contiene la matriz de costos de manera asimétrica, lo que implica que $C[i][j] \neq C[j][i]$.

2.2 Metodología del Algoritmo Evolutivo

El algoritmo evolutivo implementado sigue los siguientes pasos:

1. **Generación de la Población Inicial:**
   Se crea una población de soluciones representadas como permutaciones de ciudades. Cada individuo es una lista ordenada de índices que indican el recorrido del viajante.
2. **Evaluación del Fitness:**
   La función de fitness se define como el inverso del costo total del recorrido. Se calcula sumando los costos de viaje entre ciudades consecutivas y añadiendo el costo del regreso a la ciudad de origen. De esta forma, se transforman problemas de minimización en problemas de maximización de fitness.
3. **Selección de Padres:**
   Se utiliza el método de torneo para elegir a los individuos que se someterán a operadores genéticos. En cada torneo se selecciona el individuo con mayor fitness, favoreciendo la reproducción de soluciones de alta calidad.
4. **Operadores de Cruce y Mutación:**
   - **Cruce:** Se ha implementado el operador de cruce PMX (Partially Mapped Crossover), que garantiza que la descendencia sea una permutación válida, preservando fragmentos de los padres.

- **Mutación:** Se ha optado por el operador de mutación por inversión, que selecciona un segmento aleatorio de la solución y revierte su orden. Esto ayuda a explorar nuevas configuraciones sin alterar la validez de la permutación.

5. **Selección de Sobrevivientes:**
   Se emplea un esquema de selección *Steady-State*, en el que sólo se reemplazan de manera gradual aquellos individuos con peor fitness, asegurando que las mejores soluciones se conserven a lo largo de las generaciones.
6. **Condición de Corte:**
   El algoritmo se ejecuta hasta alcanzar un número máximo de generaciones o hasta que no se observe mejora significativa en el fitness durante un número determinado de iteraciones consecutivas.
7. **Registro de Resultados:**
   Al finalizar cada ejecución se registra en un archivo:
   - La configuración del algoritmo (parámetros y componentes utilizados).
   - La evolución del mejor fitness obtenido en cada generación.
   - La mejor solución lograda (recorrido y costo total).
   - El tiempo total de ejecución.

2.3 Configuración de Parámetros y Decisiones Tomadas

Para evaluar la efectividad y eficiencia del algoritmo, se realizaron experimentos computacionales variando los principales parámetros. En particular, se generaron todas las configuraciones posibles utilizando el producto cruzado de los siguientes conjuntos de valores:

- **Tamaño de la población:**
  population_sizes=[100,150,200]
- **Probabilidad de cruce:**
  crossover_probs=[0.7,0.8,0.9]
- **Probabilidad de mutación:**
  mutation_probs=[0.05,0.1,0.2]

- **Número máximo de generaciones:**
  max_generations=[500,1000,1500]
- Número de K para torneo
  k_value_tournament=[3, 5, 7]
- Valor de reemplazo K (steady-state)
  replacement_count=[0.2, 0.3, 0.4] (porcentaje respecto al tamaño de la población)

Al combinar todos estos valores mediante un producto cruzado, se obtuvieron un total de $3\times3\times3\times3\times3\times3=243$ configuraciones experimentales distintas. Cada configuración fue probada múltiples veces (5 repeticiones) para evaluar la estabilidad y la variabilidad de los resultados.

La elección de estas configuraciones se basó en la necesidad de balancear la exploración del espacio de soluciones (mediante una población suficientemente diversa y una probabilidad de mutación adecuada) y la explotación de las soluciones prometedoras (a través de una alta probabilidad de cruce y estrategias elitistas en la selección de sobrevivientes).

2.4 Desarrollo de Experimentos Computacionales

Se diseñaron experimentos computacionales que permitieron analizar las siguientes métricas:

- **Costo de la mejor solución:** Evaluando el fitness final obtenido.
- **Tiempo total de ejecución:** Relacionando el número de generaciones y el tamaño de la población con el tiempo de cómputo.
- **Evolución del fitness:** Registrando el mejor fitness obtenido en cada generación.

Queria probar como afectaba las distintas combinaciones de configuraciones a los algoritmos, por lo tanto quise probar realizar producto cruz entre todas las opciones que se me ocurrieron de combinaciones y automatizarlo mediante un script que hace lo siguiente:

1. Generaron todas las combinaciones de parámetros utilizando el producto cruzado de los conjuntos de valores indicados.
2. Ejecutaron el algoritmo evolutivo para cada configuración, registrando el mejor costo, el tiempo de ejecución y la evolución del fitness.
3. Almacenaron los resultados en un archivo CSV para facilitar su análisis posterior

2.5 Análisis de resultados

Luego de haber ejecutado los problemas con todas las combinaciones posibles podemos realizar un análisis de para qué valores se consiguió llegar al menor resultado para los problemas.

Problema p43

Los mejores resultados obtenidos fueron separados en un archivo CSV aparte que se utilizó para su análisis, podrá encontrarlo junto al código adjunto.

Ahora analizaremos los datos mencionados y mediante la utilización de un LLM solicite que saque conclusiones y observe relaciones. Esto es lo que encontró:

Optimización en términos de tiempo:
Configuraciones con una población moderada o alta (especialmente population_size = 200) pueden ser muy eficientes si se reduce el número máximo de generaciones (por ejemplo, 500 o 1000) y se combinan con una probabilidad de cruce de 0.8 y una probabilidad de mutación baja (0.05).

Presión selectiva y diversidad:
Utilizar un tamaño de torneo moderado (k = 3) y un porcentaje de reemplazo bajo (0.2) resulta en una presión selectiva que preserva la diversidad y acelera la convergencia, sin perder la capacidad de encontrar la solución óptima.

Trade-off entre exploración y explotación:
Aunque configuraciones con parámetros más agresivos (por ejemplo, crossover_prob = 0.9 y mutation_prob = 0.2) logran el óptimo, suelen implicar tiempos de ejecución mayores. Esto sugiere que, para este problema, no es necesario un elevado nivel de exploración si el óptimo es alcanzable con configuraciones más moderadas.

Problema br17

Este problema tiene la característica de que la mayoría de las soluciones con diferentes parámetros logran llegar a la solución óptima.

Debido a que **br17** es una instancia pequeña con un espacio de búsqueda manejable y un paisaje de fitness que favorece la convergencia, la mayoría de las combinaciones de parámetros que pruebes logran alcanzar el resultado óptimo. Esto evidencia que, en problemas de tamaño reducido, la robustez del algoritmo evolutivo y la estructura del problema permiten que incluso configuraciones con distintos niveles de agresividad en la exploración o la explotación sean efectivas. En problemas de mayor escala, sin embargo, el ajuste fino de estos parámetros suele ser mucho más crítico, ya que el espacio de búsqueda y la complejidad del paisaje pueden llevar a convergencias prematuras o a una exploración insuficiente.

Analisis de parametros

Una de las funciones en el programa, es tomar los parámetros utilizados para obtener las mejores soluciones, y volver a correr el algoritmo para ver el resultado.

Ahí podemos ver que en las ejecuciones muchas veces no volvemos a llegar al mejor resultado como anteriormente, sin embargo aun así obtenemos muy buenos resultados bastante cercanos.

La principal razón es que los algoritmos evolutivos son inherentemente estocásticos. Esto significa que, incluso utilizando exactamente los mismos parámetros, varias fuentes de aleatoriedad (como la generación de la población inicial, la selección aleatoria en torneos, la aplicación de cruces y mutaciones) influyen en la trayectoria de la búsqueda.

3. Ejemplos de resultados

A continuación se mostraran solo algunas configuraciones y soluciones, con las que se llegó al óptimo.

3.1 Problema br17

Solucion 1

Camino: [7, 16, 8, 3, 4, 6, 5, 15, 14, 11, 0, 2, 13, 1, 12, 9, 10]

Parametros

population_size	200
crossover_prob	0.7
mutation_prob	0.1
max_generations	500
k_value_tournament	3
replacement_count	0.2

Solucion 2

Camino: [16, 4, 3, 6, 5, 15, 14, 0, 11, 13, 2, 1, 10, 12, 9, 8, 7]

Parametros

population_size	200
crossover_prob	0.9
mutation_prob	0.05
max_generations	1500
k_value_tournament	5
replacement_count	0.4

3.2 Problema p43

Solucion 1

Camino: [8, 7, 6, 5, 29, 30, 27, 28, 3, 2, 1, 36, 37, 39, 38, 40, 41, 42, 26, 24, 25, 21, 22, 23, 35, 0, 4, 16, 20, 19, 18, 17, 15, 12, 13, 14, 33, 34, 31, 32, 11, 9, 10]

Parametros

population_size	100
crossover_prob	0.8
mutation_prob	0.05
max_generations	1500
k_value_tournament	3
replacement_count	0.2

Solucion 2

Camino: [32, 31, 10, 11, 8, 9, 7, 5, 6, 29, 30, 27, 28, 2, 3, 1, 36, 37, 39, 38, 40, 41, 42, 26, 24, 25, 21, 22, 23, 35, 0, 4, 17, 16, 15, 18, 19, 20, 14, 13, 12, 33, 34]

Parametros

population_size	200
crossover_prob	0.8
mutation_prob	0.2
max_generations	1000
k_value_tournament	3
replacement_count	0.4

4. Conclusiones

El desarrollo e implementación del algoritmo evolutivo para resolver el Problema del Viajante permitió comprobar que:

- **Efectividad:**
  El algoritmo es capaz de encontrar soluciones de alta calidad para instancias asimétricas del TSP. La implementación de operadores genéticos adecuados (cruce PMX y mutación por inversión) y estrategias de selección de sobrevivientes (Steady-State) garantizó la preservación de las mejores soluciones a lo largo de las generaciones.
- **Eficiencia:**
  La eficiencia del algoritmo depende de la configuración de parámetros. Los experimentos demostraron que un equilibrio adecuado entre exploración y explotación es crucial para minimizar el tiempo de ejecución sin comprometer la calidad de la solución. La utilización del producto cruzado entre diferentes valores de parámetros permitió identificar aquellas configuraciones que ofrecían un mejor compromiso entre tiempo de cómputo y costo de la solución.
- **Registro y Análisis de Resultados:**
  La incorporación de un mecanismo de registro facilitó la documentación de la evolución del fitness, la mejor solución obtenida y los tiempos de ejecución. Esto resultó fundamental para analizar y comparar el desempeño del algoritmo bajo distintas configuraciones, permitiendo ajustes y mejoras en futuras implementaciones.

En resumen, el proyecto demuestra el potencial de los algoritmos evolutivos para abordar problemas complejos como el TSP, y resalta la importancia de la experimentación sistemática y la adecuada parametrización para lograr resultados óptimos.