Práctica 1

Metaheurísticas

Curso 2021/2022

Esta primera práctica está orientada al desarrollo de técnicas de búsqueda tanto local como global. La práctica se divide en dos partes: *Hill Climbing; Simulated Annealing.* El alumno deberá realizar todo lo que se solicita en este documento, y generar un informe detallado en el que se justifiquen las respuestas y los análisis llevados a cabo. Tanto el **código nuevo generado como el informe detallado deberán entregarse por el grupo de prácticas antes de la fecha establecida**, teniendo en cuenta que esta primera práctica tiene una duración de 3 sesiones.

1. *Hill Climbing*. En este primer apartado vamos a trabajar con una búsqueda local utilizando el código Python HillClimbing.py. El código proporcionado corresponde a un algoritmo genérico de escalada por la máxima pendiente para resolver el problema del TSP. Un vecino será aquel que intercambia dos ciudades en su representación. Por ejemplo, el recorrido [0, 2, 1, 3] será vecino del [0, 1, 2, 3], pero no será vecino del [1, 3, 2, 0]. Analiza el código y comprueba que es correcto y cumple con los requisitos. A continuación, puedes utilizar el generador TSP proporcionado en el fichero TPGenerator.py para variar el número de ciudades que se utilizarán por el algoritmo *Hill Climbing.*

   Contesta a las siguientes cuestiones de manera justificada y ayudándote de gráficas cuando sea necesario:

   - ¿Cómo se comporta este algoritmo a medida que aumentamos el problema (número de ciudades en el TSP)?
   - ¿Obtiene siempre la mejor solución? ¿Por qué? ¿De qué depende?
   - Modifica el código para para comenzar la búsqueda de nuevo desde otra solución inicial (*Iterated local search*). ¿Has conseguido mejorar? ¿Por qué?

2. *Simulated Annealing.* En este segundo apartado vamos a trabajar con una búsqueda global utilizando el código Python SimAnnealing.py. El código proporcionado corresponde a un algoritmo genérico de recocido simulado para resolver el problema del TSP. El algoritmo trae por defecto una temperatura inicial de 10 y una función de enfriamiento en el que la temperatura desciende un 1% en cada iteración. El criterio de parada es una temperatura igual o inferior a 0.05. Al igual que antes, un vecino será aquel que intercambia dos ciudades en su representación. Por ejemplo, el recorrido [0, 2, 1, 3] será vecino del [0, 1, 2, 3], pero no será vecino del [1, 3, 2, 0]. Analiza el código y comprueba que es correcto y cumple con los requisitos. A continuación, puedes utilizar el generador TSP proporcionado en el fichero TPGenerator.py para variar el número de ciudades que se utilizarán por el algoritmo.

   Contesta a las siguientes cuestiones de manera justificada y ayudándote de gráficas cuando sea necesario:

- ¿Cómo se comporta este algoritmo a medida que aumentamos el problema (número de ciudades en el TSP)?
- ¿Obtiene siempre la mejor solución? ¿Por qué? ¿De qué depende?
- Analiza cómo varía el comportamiento del algoritmo a medida que cambiamos el criterio de parada y la temperatura inicial.
- Modifica el código para utilizar diferentes funciones de enfriamiento:
    - Logarítmico. To es la temperatura inicial; α es la velocidad de enfriamiento (valores menores de 1 aceleran el enfriamiento); k es la iteración en la que te encuentras.

$$T_k = \frac{\alpha \, To}{\ln (1 + k)}$$

    - Geométrico. To es la temperatura inicial; α es la velocidad de enfriamiento (valores menores de 1); k es la iteración en la que te encuentras.

$$T_k = \alpha^k \, To$$

    ¿Cómo afectan estas funciones a los resultados finales? ¿Por qué? Ayúdate representando los valores de estas funciones. Busca nuevas funciones y compáralas a las anteriores.
- ¿Cómo mejorarías el algoritmo? Por ejemplo, recalentando cada cierto tiempo. Modifica el código con esta y cualquier otra mejora que se te ocurra. Analiza los resultados.