uc3m Teoría Avanzada de la Computación

Grado en Ingeniería Informática

Curso 2021-2022

Práctica 3

Algoritmos aplicados a Optimización Combinatoria

Introducción

En esta práctica se propone estudiar algunos algoritmos que se aplican a resolver problemas de optimización combinatoria, en concreto al Problema del Viajante.

Se pide aplicar los conocimientos adquiridos en Magistral, en especial Análisis de Coste Computacional y Complejidad Computacional.

Parte Obligatoria

El problema seleccionado es el **Problema del Viajante** (Traveling Salesman Problem), uno de los 23 problemas originales identificados pertenecientes a la **clase NP-Complete** en su versión de problema de decisión.

Aquí lo vamos a tratar en su versión euclídea como problema de búsqueda para proporcionar una solución óptima, o si no es posible, subóptima.

Definición

El Problema del Viajante en su versión bidimensional euclídea consiste en una serie de ***n*** ciudades distribuidas en un plano de dos dimensiones. Todas las ciudades están interconectadas mediante carreteras.

El problema se puede representar con un grafo completo de ***n*** nodos. Los arcos entre las ciudades representan las carreteras que las conectan y están valuados, cada arco tiene un valor, concretamente la distancia euclídea entre cada par de nodos.

El problema planteado como problema de optimización consiste en encontrar un único ciclo hamiltoniano que recorra las ***n*** ciudades y que tenga un coste total mínimo (suma de los costes de cada carretera incluida en el ciclo).

Se pide:

Para abordar su resolución se van a proponer dos métodos:

1. **Búsqueda basada en Backtracking o equivalente iterativa**. Explorar la posibilidad de mejora mediante una heurística sencilla o aplicando criterios de poda. En este caso se busca garantizar siempre una solución óptima. *Será difícil encontrar soluciones para problemas de tamaños **n** mayores a 14 … 20.*

2. **Búsqueda local**. Partiendo de la solución de un algoritmo *greedy* [1], aplicar el operador *2-opt* [2]. En este caso, no se podrá garantizar una solución óptima, pero se podrá estudiar qué calidad puede proporcionar el método (en % de desviación respecto al recorrido óptimo) en un tiempo razonable.

Determinar el coste computacional de los algoritmos diseñados. Esto se deberá hacer de forma analítica del algoritmo, empírica y combinando ambos. Esto incluye siempre los desarrollos, tablas y gráficas que sirvan para apoyar vuestros análisis y conclusiones. Determinar hasta qué punto es posible obtener soluciones de calidad en función del tiempo disponible.

Para disponer de conjuntos de distribuciones de ciudades, debéis **generar vuestros propios casos de forma aleatoria**. Os puede ser útil la **librería TSPlib** que tiene alguna versión en Python y una serie de problemas de los que se conoce la solución óptima, lo cual es importante para valorar los resultados de la búsqueda local.

Para **verificar los problemas y las soluciones**, será útil disponer de algún método para imprimir un gráfico con las ciudades distribuidas en el plano y el recorrido generado que las conecta.

En web disponéis de numerosos recursos para obtener los algoritmos ya programados. Pero tened en cuenta dos cosas:

- Es frecuente que tengan errores.
- Los algoritmos deben ser sencillos. Una heurística muy elaborada, también será compleja de analizar.
- El hecho de que un programa figure cómo implementación de cierto algoritmo u heurística no significa que se le pueda atribuir la complejidad que deriva de su propuesta original.

Parte Opcional

Se pueden plantear otros estudios que se valorarán adicionalmente.

Por ejemplo:

- ¿Cómo se comporta el coste computacional de la búsqueda exacta si suprimimos arcos del grafo original? Por ejemplo, los arcos de mayor distancia para cada distancia. El estudio se basaría en analizar cómo varía el coste computacional en función de la cantidad de arcos eliminados.
- ¿Cómo se comporta el coste computacional de la búsqueda exacta si se paraleliza el algoritmo (multihilo)?
- Variantes del TSP (asimétrico, por ejemplo).

[1] En el caso de este problema, el algoritmo greedy más a mano consiste en construir el recorrido añadiendo de forma sucesiva nodos al mismo, se seleccionará en cada caso el más próximo al último nodo añadido,

[2] 2-OPT : algoritmo de búsqueda local.

De cara a la entrega:

- Incluid los estudios en una Memoria, que además explique el trabajo realizado
- Entregad el Código, y ejemplos de prueba en fichero. Es importante que os aseguréis que el código puede funcionar correctamente en cualquier ordenador. Eso quiere decir que evitéis usar instalaciones complejas de librerías, etc.
- **Se recomienda especialmente entregar el código en Notebooks para Google Colab. Estos también pueden funcionar como Jupyter Notebooks en ordenadores personales.**

EVALUACIÓN

En las prácticas se va a evaluar vuestra capacidad de abordar y resolver problemas de computación con las ideas impartidas en la asignatura. Es de suponer que los cuatro cursos del grado os habrán proporcionado las competencias necesarias para utilizar todo tipo de recursos (matemáticos, de programación, de búsqueda de información, etc.) para tener cierta soltura en esta tarea.

Se valorará en especial que apliquéis el Análisis de Coste Computacional, la obtención de las Cotas Asintóticas, y que relacionéis el trabajo con cuestiones de Complejidad Computacional.

También se evaluarán aspectos de coherencia a la hora de presentar cálculos y evaluaciones, tablas y gráficos.

Otros aspectos que se valorarán positivamente pueden ser:

- Vuestra capacidad de programar los algoritmos vosotros mismos
- Usar lenguajes más apropiados y eficientes para implementar algoritmos computacionalmente exigentes: C/C++, Cython, PyPy, Julia, Rust, en vez de Python
- La paralelización de algún algoritmo con mejoras de rendimiento.

Cualquier programa que utilicéis que no haya sido desarrollado por vosotros mismos deberá estar correctamente referenciado, igual que la bibliografía en la que os baséis.

ALGORITMOS NECESARIOS:

Algoritmo de búsqueda en profundidad: iterativo o con backtracking

- ¿Hace falta detallarlo?
- Estudiar el coste computacional empírico en base a tiempos, testigos y analítico.
- ¿Cuál es la complejidad computacional del problema con este algoritmo?
- Suponed que todas las instancias son igual de costosas. ¿Es esto así?
- Programad alguna heurística o poda sencilla (es importante que sea sencilla, y que garantice la solución óptima).
    a. Guiar la búsqueda empezando por conectar siempre el nodo más próximo

b. Podando aquellas búsquedas en profundidad que tengan una longitud parcial mayor a la mejor obtenida hasta el momento

- ¿Qué aportan exactamente estas heurísticas?

Algoritmo de búsqueda local:

- Generar una primera solución (mediocre) con un algoritmo *greedy*.
  - Cada ciudad se conecta a la más próxima de las disponibles (sin conectar)
- Aplicar 2-Opt de forma reiterada a la solución parcial

  Bucle externo:

    Bucle medio: para cada par de ciudades consecutivas en el recorrido AB

      Bucle interno: Para cada par de ciudades consecutivas en el recorrido CD diferentes de AB:

        Desconectar A de B y C de D

        Reconectar A con C y B con D

        Ojo, debe generar un único ciclo, no dos.

        Evaluar longitud del recorrido (*)

        Si nuevo recorrido es más corto ==> aceptarlo como nueva solución

        Si nuevo recorrido es más largo ==> rechazarlo y retomar solución anterior

      Bucle interno - Fin

    Bucle medio - Fin

  Bucle externo Fin: Aplicar bucles medio e interno mientras haya cambios en el recorrido.

- Estudiar el coste computacional empírico en base a tiempos, testigos, y analítico.
- ¿Cuál es la complejidad computacional del problema con este algoritmo?
- ¿Qué calidad es posible lograr con estas heurísticas?

(*) Si representamos el recorrido de ciudades mediante un vector en el que cada número representa una ciudad:

```
9 0 4 5 3 1 2 8 6 7
    A B       C D
```

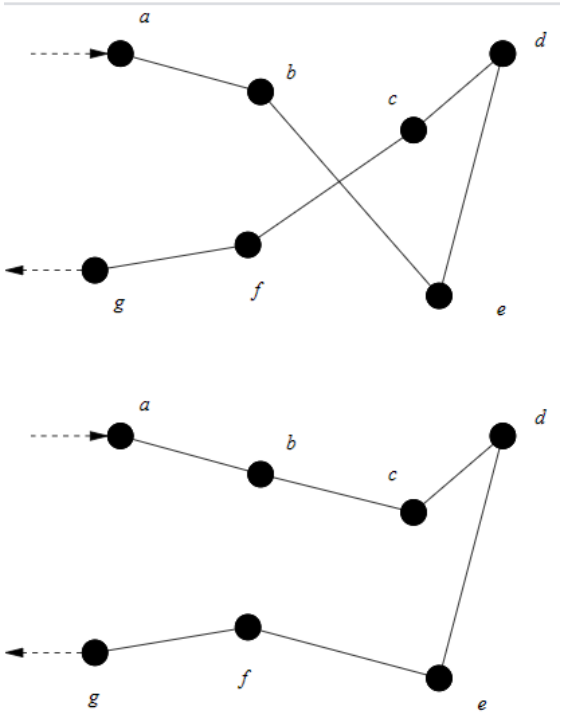
El proceso de reconectar los pares AB y CD consiste en invertir la cadena BC

```
9 0 4 8 2 1 3 5 6 7
    A C       B D
```

Ojo si hay que invertir la cadena exterior…

Se puede comprobar que no hace falta recalcular la longitud de todo el recorrido. Basta con comprobar si |AB|+[CD| > [AC|+|BD| . ¡Comprobadlo vosotros!

Es más, se puede evaluar esta comprobación antes de decidir si reconectar las ciudades.



By PierreSelim - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=8044684

.