UNIVERSIDAD
DE GRANADA

Inteligencia Computacional

Práctica de algoritmos evolutivos

Problemas de optimización combinatoria
QAP

Curso 2020-2021

Máster en Ingeniería Informática

Departamento de Ciencias de la Computación e Inteligencia Artificial

Práctica 2

QAP

El objetivo de esta práctica es resolver un problema de optimización típico utilizando técnicas de computación evolutiva. Deberá implementar varias variantes de algoritmos evolutivos para resolver el problema de la asignación cuadrática, incluyendo como mínimo las variantes que se describen en este guión de prácticas.

1 El problema de la asignación cuadrática

El problema de la asignación cuadrática o QAP [Quadratic Assignment Problem] es un problema fundamental de optimización combinatoria con numerosas aplicaciones. El problema se puede describir de la siguiente forma:

Supongamos que queremos decidir dónde construir n instalaciones (p.ej. fábricas) y tenemos n posibles localizaciones en las que podemos construir dichas instalaciones. Conocemos las distancias que hay entre cada par de instalaciones y también el flujo de materiales que ha de existir entre las distintas instalaciones (p.ej. la cantidad de suministros que deben transportarse de una fábrica a otra). El problema consiste en decidir dónde construir cada instalación de forma que se minimize el coste de transporte de materiales.

Formalmente, si llamamos $d(i, j)$ a la distancia de la localización i a la localización j y $w(i, j)$ al peso asociado al flujo de materiales que ha de transportarse de la instalación i a la instalación j, hemos de encontrar la asignación de instalaciones a localizaciones que minimice la función de coste

$$\sum_{i,j} w(i,j)\ d(p(i),p(j))$$

donde p() define una permutación sobre el conjunto de instalaciones.

Igual que en el problema del viajante de comercio, que se puede considerar un caso particular del QAP, una solución para el problema es una permutación del conjunto de instalaciones que indica dónde se debe construir cada una.

> NOTA: El problema del viajante de comercio o TSP [Traveling Salesman Problem] puede interpretarse como un caso particular del QAP si se asume que los flujos conectan todas las instalaciones formando única y exclusivamente

un anillo (teniendo todos los flujos el mismo peso, una constante distinta de cero). Otros problemas de optimización combinatoria pueden plantearse de forma similar.

El problema de la asignación cuadrática es un problema habitual en Investigación Operativa y, además de utilizarse para decidir la ubicación de plantas de producción, también se puede utilizar como modelo para colocar los componentes electrónicos de un circuito sobre una placa impresa o los módulos de un circuito integrado en la superficie de un microchip.

Por su interés teórico y práctico, existe una variedad muy amplia de algoritmos que abordan la resolución del problema de la asignación cuadrática. Al ser un problema NP-completo, el diseño y aplicación de algoritmos exactos para su resolución no es viable cuando n es grande. Nos centraremos, por tanto, en el diseño de algoritmos evolutivos y evaluaremos su rendimiento sobre instancias concretas del problema.

2 Implementación y análisis de resultados

- Implemente un algoritmo genético estándar que resuelva el problema de la asignación cuadrática. Además de su implementación y una descripción del algoritmo genético utilizado (técnica de representación, mecanismo de selección, operadores de cruce y mutación...), debe proporcionar la permutación obtenida y el coste asociado a la misma para los distintos conjuntos de datos proporcionados. No olvide incluir los parámetros concretos utilizados en la ejecución de su algoritmo genético (tamaño de la población, número de generaciones necesario para obtener la solución encontrada, probabilidades de cruce y mutación, etc.).

- Implemente una variante baldwiniana de su algoritmo genético estándar, que incorpore técnicas de optimización local (p.ej. heurísticas greedy) para dotar a los individuos de su población de capacidad de "aprendizaje". Compare los resultados obtenidos con respecto a los que obtuvo el algoritmo genético estándar.

  NOTA: En la variante baldwiniana del algoritmo genético, para evaluar el fitness de cada individuo, se utiliza dicho individuo como punto inicial de una búsqueda local (p.ej. ascensión de colinas por la máxima pendiente) hasta que se alcanza un óptimo local. El valor de ese óptimo local determina el fitness del individuo. Sin embargo, a la hora de formar descendientes, se utiliza el material genético del individuo original (sin incluir las mejoras "aprendidas" al aplicar la técnica de búsqueda local).

- Implemente una variante lamarckiana de su algoritmo genético estándar, que incorpore técnicas de optimización local (p.ej. heurísticas greedy) para dotar a los individuos de su población de capacidad de "aprendizaje" y permita que lo "aprendido" se pueda heredar directamente de padres a hijos. Compare los resultados obtenidos con respecto a los que obtuvo con las variantes anteriores.

NOTA: En la variante lamarckiana del algoritmo genético, el fitness de los individuos se evalúa igual que en la variante baldwiniana, si bien ahora los descendientes de un individuo se forman a partir de la solución mejorada que se consigue utilizando técnicas de búsqueda local (esto es, los descendientes heredan los rasgos adquiridos por sus padres en su proceso de "aprendizaje").

- Realice un estudio comparativo de las tres variantes implementadas utilizando para ello los conjuntos de datos de prueba que se proporcionan con este guión. Visualice gráficamente los resultados de las distintas ejecuciones mostrando la evolución del fitness de la mejor solución encontrada en cada generación. ¿Cómo se comportan las distintas variantes en cuanto a la calidad de las soluciones obtenidas y al tiempo necesario para obtenerlas?

2.1 Documentación y entrega de la práctica

- Ejecute su implementación sobre el conjunto de datos `tai256c` y envíe los resultados que haya obtenido a través de la página web habilitada al efecto (`http://goo.gl/p1UZlp`). El envío de resultados debe realizarlo utilizando la misma dirección de correo con la que aparezca registrado en la asignatura y ha de incluir los parámetros del algoritmo genético utilizado, el coste de la mejor solución obtenida y la permutación que da lugar a ese valor, representada como una secuencia de números enteros separados por espacios. Puede enviar sus resultados tantas veces como desee.

- Elabore una memoria en la que se recoja la experimentación realizada durante la elaboración de esta práctica y los gráficos oportunos. La memoria deberá entregarse en formato PDF e irá acompañada de todos los ficheros correspondientes a las distintas implementaciones efectuadas. La entrega de la memoria en PDF y de los ficheros de código en un ZIP deberá realizarse a través de la plataforma docente de la asignatura (Google Classroom) antes del **12 de febrero de 2021 a las 23:59**.

2.2 Evaluación de la práctica

- 30 % por la implementación del algoritmo genético básico, su variante baldwiniana y su variante lamarckiana.

- 20 % por la memoria de la práctica (documentación de los experimentos y pruebas realizadas junto con su análisis de resultados).

- 50 % por los resultados obtenidos, usando como base el mejor resultado conocido para el problema, 44759294. La puntuación correspondiente a este apartado se calculará en función de la diferencia del coste obtenido (c) con el coste de la mejor solución conocida (m), en porcentaje: $max\{5 - 100 * (c - m)/m, 0\}$.

3 Datos de prueba

Los casos de prueba para comprobar el funcionamiento de las distintas heurísticas greedy están disponibles en la plataforma de docencia de la asignatura y se han obtenido de la biblioteca QAPLIB (`http://www.seas.upenn.edu/qaplib/`).El formato de los ficheros de datos es el siguiente:

```
n
A
B
```

donde n es el tamaño del problema, mientras que A y B son las matrices de flujos y distancias, indistintamente (el problema es simétrico, por lo que no influye en el resultado cuál es cuál).

4 Heurísticas greedy para el problema de la asignación cuadrática

Para problemas NP, los algoritmos greedy no propocionan soluciones óptimas, pero pueden sernos útiles como técnicas de optimización local para obtener soluciones aceptables de forma muy eficiente.

A la hora de diseñar algoritmos greedy que resuelvan un problema de tipo combinatorio, como el de la asignación cuadrática, probaremos con distintos tipos de estrategias heurísticas, de forma que podamos evaluar de forma empírica cuáles funcionan mejor en la práctica:

- Estrategias constructivas.
- Estrategias de transposición.

Para el primer tipo de estrategia, utilizaremos una variante de la heurística del *vecino más cercano*, cuyo funcionamiento es extremadamente simple: dada la localización $p(i)$ de una instalación i, a continuación se escoge la instalación j con la i tiene un mayor flujo asociado $w(i, j)$ y se construye en la localización $p(j)$ que esté más cerca del lugar $p(i)$ donde hemos situado la instalación i. En este tipo de algoritmos, podemos considerar, una por una, distintas combinaciones de punto de partida y devolver el mejor resultado obtenido en las distintas ejecuciones realizadas.

En las estrategias de transposición, la idea es comenzar con una permutación inicial e ir intercambiando las posiciones de dos instalaciones mediante algún criterio de tipo greedy. Para poder implementar este tipo de estrategia, han de definirse dos elementos:

1. Cómo se construye la permutación inicial.
2. Qué transposiciones se consideran en cada momento, p.ej. 2-opt ó 3-opt.

Además, deberemos decidir cuántas transposiciones permitiremos realizar antes de que el algoritmo greedy devuelva un resultado. Usando un algoritmo greedy, si disminuye el coste asociado a la solución al realizar un intercambio de posición, entonces mantenemos el intercambio. Si no, el intercambio no se efectúa y seguimos con la solución que ya teníamos.

En pseudocódigo, un algoritmo greedy basado en 2-opt para el problema QAP podría tener el siguiente aspecto:

```
1  S = candidato inicial con coste c(S)
2
3  do {
4
5    mejor = S
6
7    for i=1..n
8        for j=i+1..n
9            T = S tras intercambiar i con j
10           if c(T) < c(S)
11              S = T
12
13 } while (S != mejor)
```

> NOTA: El algoritmo 2-opt asociado al problema consideraría $n(n-1)/2$ intercambios antes de modificar la mejor solución actual S y, en cada iteración, realizaría un único intercambio (aquél con el que se obtuviese un coste menor de entre todas las transposiciones). El algoritmo greedy hace el intercambio permanente en cuanto encuentra una mejora, lo que le permite ser más eficiente a costa de encontrar peores soluciones que el algoritmo 2-opt.

Algoritmos greedy como los descritos se pueden incorporar a un algoritmo evolutivo, ya sea para establecer la población inicial (p.ej. estrategias constructivas) o para dotar de capacidad de "aprendizaje" a los individuos de la población (p.ej. estrategias de transposición). Usando las permutaciones representadas por los individuos de la población de un algoritmo evolutivo como punto de partida, podemos implementar fácilmente las variantes baldwiniana y lamarckiana de un algoritmo genético estándar si recurrimos al uso de heurísticas greedy.