UNIVERSIDAD DE GRANADA

E.T.S. DE INGENIERÍAS INFORMÁTICA y DE TELECOMUNICACIÓN

Departamento de Ciencias de la Computación e Inteligencia Artificial

Algorítmica

Guión de Prácticas

Práctica 4: Algoritmos de Vuelta Atrás (Backtracking) y de Ramificación y Poda (Branch and Bound)

Curso 2018-2019

Grado en Informática

1. Objetivo

El objetivo de esta práctica es que el estudiante aprecie la potencia de los métodos vuelta atrás para resolver problemas, pero también comprenda sus limitaciones. Para ello cada equipo de estudiantes deberá resolver uno de los problemas (asignado al azar) que se describen en la sección 2, así como exponer y defender su propuesta en clase. En todos los casos se debe especificar claramente, además del algoritmo, la representación del problema, la representación de la solución, las restricciones explícitas e implícitas, así como las posibles cotas a utilizar.

Adicionalmente, todos los equipos deberán implementar (y exponer y defender) algoritmos de ramificación y poda para el problema del viajante de comercio siguiendo las indicaciones descritas en la sección 3.

2. Problemas

2.1. Laberinto

El problema consiste en encontrar la salida de un laberinto. Más concretamente, supondremos que el laberinto se representa mediante una matriz bidimensional de tamaño $n \times n$. Cada posición almacena un valor 0 si la casilla es transitable y cualquier otro valor si la casilla no es transitable. Los movimientos permitidos son a casillas adyacentes de la misma fila o la misma columna. Podemos suponer que las casillas de entrada y salida del laberinto son (0,0) y $(n-1,n-1)$ respectivamente. Por tanto el problema consiste en, dada una matriz que representa el laberinto, encontrar si existe un camino para ir desde la entrada hasta la salida. Diseñar e implementar un algoritmo vuelta atrás para resolver el problema. Modificar el algoritmo para que encuentre el camino más corto. Realizar un estudio empírico de la eficiencia de los algoritmos.

2.2. Cena de gala

Se va a celebrar una cena de gala a la que asistirán n invitados. Todos se van a sentar alrededor de una única gran mesa rectangular, de forma que cada invitado tendrá sentados junto a él a otros dos comensales (uno a su izquierda y otro a su derecha). En función de las características de cada invitado (por ejemplo categoría o puesto, lugar de procedencia,...) existen unas normas de protocolo que indican el nivel de conveniencia de que dos invitados se sienten en lugares contiguos (supondremos que dicho nivel es un número entero entre 0 y 100). El nivel de conveniencia total de una asignación de invitados a su puesto en la mesa es la suma de todos los niveles de conveniencia de cada invitado con cada uno de los dos invitados sentados a su lado. Se desea sentar a los invitados de forma que el nivel de conveniencia global sea lo mayor posible. Diseñar e implementar un algoritmo vuelta atrás para resolver este problema. Realizar un estudio empírico de su eficiencia.

2.3. Cuadrado Latino

Un cuadrado latino consiste en un tablero o matriz de tamaño $n \times n$ que hay que rellenar con los numeros de 1 a n sin que se repitan números en ninguna fila ni en ninguna columna.

Por ejemplo, en el caso 3×3, un posible cuadrado latino de ese tamaño es:

	0	1	2
0	1	2	3
1	2	3	1
2	3	1	2

Diseñad e implementad un algoritmo vuelta atrás para construir cuadrados latinos de tamaño $n \times n$ (para generar solo uno o todos los posibles). Modificad el algoritmo para que los cuadrados latinos que genere cumplan un conjunto de restricciones (una asignación previa de valores en determinadas posiciones de la matriz). Realizar un estudio empírico de su eficiencia.

2.4. Pares de Estudiantes

Supongamos que tenemos n estudiantes en una clase y queremos crear con ellos equipos formados por parejas (podemos suponer que n es un número par). Se dispone de una matriz p de tamaño $n \times n$ en la que $p(i, j)$ indica el nivel de preferencia que el estudiante i tiene para trabajar con el estudiante j. El valor del emparejamiento del estudiante i con el j es $p(i, j) * p(j, i)$. Se trata de encontrar un emparejamiento para todos los estudiantes de forma que se maximice la suma de los valores de los emparejamientos.

Diseñad e implementad un algoritmo de vuelta atrás para resolver el problema. Realizar un estudio empírico de su eficiencia.

2.5. Recipientes

Tenemos n objetos de pesos $w_1, w_2, \ldots, w_n$, y un número ilimitado de recipientes iguales con capacidad máxima R (siendo $w_i \leq R$, $\forall i$). Los objetos se deben meter en los recipientes sin partirlos, y sin superar su capacidad máxima. Se busca el mínimo número de recipientes necesarios para colocar todos los objetos.

Diseñad e implementad un algoritmo de vuelta atrás para encontrar la solución óptima del problema. Realizar un estudio empírico de su eficiencia.

3. El problema del viajante de comercio

El problema del viajante de comercio ya se ha comentado y utilizado en la práctica sobre algoritmos voraces, donde se estudiaron métodos de este tipo para encontrar soluciones razonables (no óptimas necesariamente) a este problema. Si se desea encontrar una solución óptima es necesario utilizar métodos más potentes (y costosos), como la vuelta atrás y la ramificación y poda, que exploren el espacio de posibles soluciones de forma más exhaustiva.

Así, un algoritmo de vuelta atrás comenzaría en la ciudad 1 (podemos suponer sin pérdida de generalidad, al tratarse de encontrar un tour, que la ciudad de inicio y fin es esa ciudad) e intentaría incluir como parte del tour la siguiente ciudad aún no visitada, continuando de este modo hasta completar un tour. Para agilizar la búsqueda de la solución se deben considerar como ciudades válidas para una posición (ciudad actual) sólo aquellas que satisfagan las restricciones del problema (en este caso ciudades que aún no hayan sido visitadas). Cuando para un nivel

no queden más ciudades válidas, el algoritmo hace una vuelta atrás proponiendo una nueva ciudad válida para el nivel anterior.

Para emplear un algoritmo de ramificación y poda es necesario utilizar una cota inferior: un valor menor o igual que el verdadero coste de la mejor solución (la de menor coste) que se puede obtener a partir de la solución parcial en la que nos encontremos.

Una posible alternativa sería la siguiente: como sabemos cuáles son las ciudades que faltan por visitar, una estimación optimista del costo que aún nos queda será, para cada ciudad, el coste del mejor (menor) arco saliente de esa ciudad. La suma de los costes de esos arcos, más el coste del camino ya acumulado, es una cota inferior en el sentido antes descrito.

Para realizar la poda, guardamos en todo momento en una variable C el costo de la mejor solución obtenida hasta ahora (que se utiliza como cota superior global: la solución óptima debe tener un coste menor o igual a esa). Esa variable puede inicializarse con el costo de la solución obtenida utilizando un algoritmo voraz (como los utilizados en la práctica 2). Si para una solución parcial, su cota inferior es mayor que C entonces se puede realizar la poda.

Como criterio para seleccionar el siguiente nodo que hay que expandir del árbol de búsqueda (la solución parcial que tratamos de expandir), se empleará el criterio LC o "más prometedor". En este caso consideraremos como nodo más prometedor aquel que presente el menor valor de cota inferior. Para ello se debe de utilizar una cola con prioridad que almacene los nodos ya generados (nodos vivos).

Además de devolver el costo de la solución encontrada (y en su caso el tour correspondiente), se deben de obtener también resultados relativos a complejidad: número de nodos expandidos, tamaño máximo de la cola con prioridad de nodos vivos, número de veces que se realiza la poda y el tiempo empleado en resolver el problema.

Las pruebas del algoritmo pueden realizarse con los mismos datos empleados en la práctica 2 (teniendo en cuenta que el tamaño de problemas que se pueden abordar con estas técnicas es mucho más reducido que con los métodos voraces). La visualización de las soluciones también puede hacerse de la misma forma que en la práctica 2 (usando gnuplot).

3.1. Tareas a realizar

Construir un programa que utilice la técnica de ramificación y acotación para resolver el problema del viajante de comercio en las condiciones descritas anteriormente, empleando la función de acotación comentada, o alguna otra.

Opcionalmente, construir también un programa que utilice vuelta atrás, pero utilizando también la función de acotación descrita anteriormente, y realizar un estudio experimental comparativo con el algoritmo de ramificación y poda.