Práctica 4. Algoritmos Backtracking - Cena de Gala

Noelia Escalera Mejías Alejandro Menor Molinero
Javier Núñez Suárez Adra Sánchez Ruiz
Jesús Torres Sánchez

May 15, 2019

1 Descripción del problema

Podemos describir el problema de la Cena de Gala de la siguiente forma:

Se va a realizar una cela de gala a la que van a asistir n invitados. Cada invitado va a tener sentados junto a él a dos comensales (uno a su izquierda y otro a su derecha). Dependiendo de las características del invitado, existe una cierta conveniencia de que dos invitados se sienten juntos, definiendo la conveniencia global de la mesa como la suma de todos los niveles de conveniencia entre cada pareja de invitados sentados de forma contigua. El objetivo principal es conseguir sentar a los invitados de forma que el nivel de conveniencia global sea el máximo.

Por tanto, nuestro algoritmo debe ser capaz de encontrar la asignación de invitados a partir de la cual podamos maximizar la convenienca global.

2 Representación de la solución

A la hora de representar el problema, hemos usado un TDA *Solucion*, que almacena la solución y otra información relevante de la clase, además de todas las operaciones básicas para la gestión las soluciones (necesarias para el algoritmo Backtracking):

- **Información almacenada en el TDA** *Solucion*:
  - x: almacena la solución que se va generando en cada momento. Se representa por medio de un vector de enteros (conceptualmente es un vector circular), en el que cada componente está asociada a un comensal.

- *afinidades*: se representan como una matriz, en la que cada elemento m[i][j] tiene asociado la conveniencia de sentar juntos al invitado i con el invitado j. Por tanto, deben coincidir m[i][j] con m[j][i].
- n: contiene el número de comensales.
- *solucionOptima*: almacena la mejor solución encontrada hasta el momento, es decir, la que tiene una mayor afinidad global.
- *comensalesYaSentados*: representado por un vector de booleanos, almacena la información de qué comensales están ya sentados. Este vector será necesario a la hora de comprobar la factibilidad.
- *afinidadActual*: contiene la afinidad acumulada para los comensales seleccionados. Esta variable es necesaria para el cálculo y comprobaciones de cotas.
- *Afinidades optimistas*: se representa por un vector que tiene asociado, para cada comensal, una estimación optimista de su afinidad.
- *AfinidadAcabarMesa*: esta variable almacena una afinidad optimista entre el primer asiento seleccionado y el último. Es necesaria para que, al determinar el cálculo de las cotas locales, al ser al mesa un vector circular, poder cerrar el ciclo.

- **Operaciones del TDA Solucion**:
  - *Constructor*: realiza una serie de operaciones:
    - Inicializa las variables del TDA a sus valores por defecto.
    - Inicializamos *solucionOptima* y $maximoValor$ a partir de la versión Greedy (no óptima) del algoritmo.
    - Calcula las $afinidadesOptimistas$ y la $afinidadAcabarMesa$ para evitar hacerlo posteriormente en el cálculo de las cotas locales. El proceso de estimación es bastante sencillo: para cada comensal, buscamos las dos mayores afinidades que tenga y le asociamos el promedio de ambas.
  - *hemosTerminado(k)*: devuelve si ya hemos recorrido el total de comensales ($k == n$).
  - *iniaComp(k)*: inicializamos la componente k al valor NULO, que en nuestro caso es 0.
  - *sigValComp(k)*: asignamos a la componente k de la solución actual el siguiente valor ($x[k] + +$).
  - *factible(k)*: para determinar su factibilidad, debemos comprobar si el comensal $x[k]$ ya estaba sentado en otro sitio, y si la cota local más la afinidad actual (afinidad entre el comensal $x[k]$ y el anterior) es mayor que la cota global. Debemos controlar la correcta actualización de las variables.
  - *todosGenerados(k)*: devuelse si se han generado todos los posibles comensales para $x[k], (x[k] == n)$.

- *procesaSolucion*: actualizamos la afinidad actual, y la cota global y la solución óptima en el caso de que la afinidad actual sea mayor que la global.
- *imprimeSolucion*: devuelve la información asoicada a las afinidades, y la solución óptima.
- *vueltaAtras(k)*: restaura los valores de la afinidadActual y de comensalesYaSentados (necesario para cuando damos una vuelta atrás en el algoritmo de Backtracking).
- *cotaLocal(k)*: genera la cota local a partir de k. Para ello, obtiene la suma de las afinidadesOptimistas asociadas a los comensales ya sentados más la afinidadAcabarMesa para cerrar el ciclo.

3 Algoritmo Backtracking

El algoritmo sigue el esquema básico de los algoritmos Backtracking vistos en clase: *INSERTAR ESQUEMA ALGORITMO BACKTRACKING*

4 Descripción de las cotas usadas

En nuestra solución, hemos usado dos tipos de cotas: una **cota global**, que determina la máxima afinidad global encontrada hasta el momento. Si aún no se ha encontrado ninguna solución, estará inicializada de acuerdo a una versión Greedy del algoritmo. Por otro lado, también necesitamos una **cota local** (tal y como se ha mencionado anteriormente), necesaria para poder podar una rama en el caso de que la cota local (cota optimista) no supere a la cota global. La poda se puede realizar debido a que, si una cota local optimista genera peores resultados que una solución ya encontrada, esa rama no va a contener una solución mejor, por lo que no tiene sentido recorrerla.

5 Estudio empírico

6 Conclusiones