DISEÑO Y ANÁLISIS DE ALGORITMOS

Ramificación y poda

Maximum diversity problem

J. Marcos Moreno-Vega
Christopher Expósito Izquierdo

Objetivo:

Proponer, implementar y evaluar algoritmos heurísticos y de ramificación y poda para el *Maximum diversity problem*.

Tareas:

Además de las tareas descritas en el presente documento, los alumnos tendrán que realizar las modificaciones que se planteen durante la corrección de la práctica. Asimismo, tendrán que responder a un cuestionario de preguntas tipo test sobre los contenidos teóricos de los algoritmos constructivos y búsquedas por entornos.

Corrección:

19 de mayo de 2015

Evaluación:

Código fuente y memoria: hasta 5 puntos; si el día de la corrección falta algún código o este es incorrecto, la práctica se calificará como No apta.

Modificación propuesta el día de la corrección: hasta 3 puntos.

Cuestionario sobre los contenidos teóricos: hasta 2 puntos.

Lenguaje de programación:

Java o C++.

Maximum diversity problem

En el *Maximum diversity problem* se desea encontrar el subconjunto de elementos de diversidad máxima de un conjunto dado de elementos.

Sea dado un conjunto $\mathbb{S} = \{s_1, s_2, \ldots, s_n\}$ de n elementos, en el que cada elemento s_i es un vector $s_i = (s_{i1}, s_{i2}, \ldots, s_{iK})$. Sea, asimismo, d_{ij} la distancia entre los elementos i y j. Si $m < n$ es el tamaño del subconjunto que se busca el problema puede formularse como:

$$\text{Maximizar } z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

sujeto a:

$$\sum_{i=1}^{n} x_i = m$$
$$x_i \in \{0, 1\} \qquad i = 1, 2, \ldots, n$$

donde:

$$x_i = \begin{cases} 1 & \text{si } s_i \text{ pertenece a la solución} \\ 0 & \text{en caso contrario} \end{cases}$$

La distancia d_{ij} depende de la aplicación real considerada. En muchas aplicaciones se usa la distancia euclídea. Así:

$$d_{ij} = d(s_i, s_j) = \sqrt{\sum_{r=1}^{K} (s_{ir} - s_{jr})^2}$$

Un constructivo voraz

El centro de gravedad de un conjunto de elementos $X = \{s_i : i \in I\}$ con $I \subseteq \{1, 2, \ldots, n\}$ se define como

$$centro(X) = \frac{1}{|X|}\left(\sum_{i \in I} s_{i1}, \sum_{i \in I} s_{i2}, \ldots, \sum_{i \in I} s_{iK}\right)$$

Un algoritmo constructivo voraz para el *Maximum diversity problem* parte del subconjunto S formado por el elemento más alejado del centro de gravedad de $\mathbb{S}$. A continuación, añade iterativamente a este subconjunto el elemento más alejado de su centro de gravedad hasta que S tenga m elementos. El pseudocódigo de este algoritmo se muestra en la figura 1.

Algoritmo constructivo voraz

1: $Elem = \mathbb{S}$;
2: $S = \emptyset$;
3: Obtener $s_c = centro(Elem)$;
4: **repeat**
5: Obtener el elemento $s_* \in Elem$ más alejado de s_c;
6: $S = S \cup \{s_*\}$;
7: $Elem = Elem - \{s_*\}$;
8: Obtener $s_c = centro(S)$;
9: **until** $(|S| = m)$
10: Devolver S;

Figura 1: Algoritmo constructivo voraz

Implementación

Las instancias del problema se suministrarán en un fichero de texto con el siguiente formato: en las dos primeras filas se encuentran, respectivamente, el número de elementos (n) y la dimensión (K) de los mismos; a continuación, se enumeran las coordenadas de cada elemento (una fila por elemento).

Por ejemplo, si $n = 4$ y $K = 3$, un fichero de texto para el problema podría contener los siguiente datos:

```
4
3
4,93  9,83  3,16
2,75  0,64  2,59
3,70  2,02  9,35
0,15  8,60  9,67
```

Tareas

1. Algoritmo constructivo voraz

   (a) Diseñar e implementar el algoritmo constructivo voraz descrito en la figura 1.

   (b) Completar la tabla de resultados 1.

2. Nuevo algoritmo voraz

   (a) Diseñar e implementar un nuevo algoritmo voraz, no necesariamente constructivo, para el problema.

   (b) Completar la tabla de resultados 1 para este nuevo algoritmo.

3. Búsqueda local

   (a) Diseñar e implementar un algoritmo de búsqueda local para el problema.

   (b) Completar la tabla de resultados 1 para esta búsqueda local.

4. Grasp

   (a) Diseñar e implementar un GRASP para el problema.

   (b) Completar las tablas de resultados 2 y 3.

   (c) Enumerar las conclusiones que se extraen de los resultados computacionales.

5. Ramificación y Poda

   (a) Diseñar e implementar un algoritmo de ramificación y poda para el problema.

   (b) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo constructivo voraz descrito en la figura 1. Usar como estrategia de ramificación la que expande el nodo con cota superior más pequeña.

   (c) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo constructivo que has propuesto. Usar como estrategia de ramificación la que expande el nodo con cota superior más pequeña.

   (d) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo GRASP. Usar como estrategia de ramificación la que expande el nodo con cota superior más pequeña.

   (e) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo constructivo voraz descrito en la figura 1. Usar como estrategia de ramificación la que expande el nodo más profundo (es decir, siguiendo una estrategia de búsqueda en profundidad).

(f) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo constructivo que has propuesto. Usar como estrategia de ramificación la que expande el nodo más profundo (es decir, siguiendo una estrategia de búsqueda en profundidad).

(g) Completar la tabla de resultados 4 usando como cota inferior la suministrada por el algoritmo GRASP. Usar como estrategia de ramificación la que expande el nodo más profundo (es decir, siguiendo una estrategia de búsqueda en profundidad).

(h) Enumerar las conclusiones que se extraen de los resultados computacionales.

Qué debe presentar el alumno

a) Código fuente, debidamente comentado, y fichero ejecutable.

b) Una memoria en formato pdf en la que se describan brevemente los elementos necesarios para comprender los diseños propuestos. Entre otros, hay que describir la estructura de datos que almacena las soluciones del problema, la estructura de entornos usada en la búsqueda local, cómo se construye la lista restringida de candidatos del algoritmo GRASP y cómo se obtienen las cotas del algoritmo de ramificación y poda.

c) Las tablas de resultados descritas en el presente documento y las conclusiones que se obtienen de dichos resultados.

Bibliografía

Martí, R., Gallego, M., Duarte, A. A branch and bound algorithm for the maximum diversity problem. European Journal of Operational Research, 200 (2010) pp. 36-44

Algoritmo ...

Problema	n	K	m	z	S	CPU
*max_div*_15_2	15	2	2			
*max_div*_15_2	15	2	3			
*max_div*_15_2	15	2	4			
*max_div*_15_2	15	2	5			
*max_div*_20_2	20	2	2			
*max_div*_20_2	20	2	3			
*max_div*_20_2	20	2	4			
*max_div*_20_2	20	2	5			
*max_div*_30_2	30	2	2			
*max_div*_30_2	30	2	3			
*max_div*_30_2	30	2	4			
*max_div*_30_2	30	2	5			
*max_div*_15_3	15	3	2			
*max_div*_15_3	15	3	3			
*max_div*_15_3	15	3	4			
*max_div*_15_3	15	3	5			
*max_div*_20_3	20	3	2			
*max_div*_20_3	20	3	3			
*max_div*_20_3	20	3	4			
*max_div*_20_3	20	3	5			
*max_div*_30_3	30	3	2			
*max_div*_30_3	30	3	3			
*max_div*_30_3	30	3	4			
*max_div*_30_3	30	3	5			

Tabla 1: Tabla de resultados

Algoritmo GRASP

Problema	n	K	m	*Iter*	$\|LRC\|$	z	S	*CPU*
*max_div*_15_2	15	2	2	10	2			
*max_div*_15_2	15	2	2	10	3			
*max_div*_15_2	15	2	2	20	2			
*max_div*_15_2	15	2	2	20	3			
*max_div*_15_2	15	2	3	10	2			
*max_div*_15_2	15	2	3	10	3			
*max_div*_15_2	15	2	3	20	2			
*max_div*_15_2	15	2	3	20	3			
*max_div*_15_2	15	2	4	10	2			
*max_div*_15_2	15	2	4	10	3			
*max_div*_15_2	15	2	4	20	2			
*max_div*_15_2	15	2	4	20	3			
*max_div*_15_2	15	2	5	10	2			
*max_div*_15_2	15	2	5	10	3			
*max_div*_15_2	15	2	5	20	2			
*max_div*_15_2	15	2	5	20	3			
*max_div*_20_2	20	2	2	10	2			
*max_div*_20_2	20	2	2	10	3			
*max_div*_20_2	20	2	2	20	2			
*max_div*_20_2	20	2	2	20	3			
*max_div*_20_2	20	2	3	10	2			
*max_div*_20_2	20	2	3	10	3			
*max_div*_20_2	20	2	3	20	2			
*max_div*_20_2	20	2	3	20	3			
*max_div*_20_2	20	2	4	10	2			
*max_div*_20_2	20	2	4	10	3			
*max_div*_20_2	20	2	4	20	2			
*max_div*_20_2	20	2	4	20	3			
*max_div*_20_2	20	2	5	10	2			
*max_div*_20_2	20	2	5	10	3			
*max_div*_20_2	20	2	5	20	2			
*max_div*_20_2	20	2	5	20	3			
*max_div*_30_2	20	2	2	10	2			
*max_div*_30_2	20	2	2	10	3			
*max_div*_30_2	20	2	2	20	2			
*max_div*_30_2	20	2	2	20	3			
*max_div*_30_2	20	2	3	10	2			
*max_div*_30_2	20	2	3	10	3			
*max_div*_30_2	20	2	3	20	2			
*max_div*_30_2	20	2	3	20	3			
*max_div*_30_2	20	2	4	10	2			
*max_div*_30_2	20	2	4	10	3			
*max_div*_30_2	20	2	4	20	2			
*max_div*_30_2	20	2	4	20	3			
*max_div*_30_2	20	2	5	10	2			
*max_div*_30_2	20	2	5	10	3			
*max_div*_30_2	20	2	5	20	2			
*max_div*_30_2	20	2	5	20	3			

Tabla 2: Tabla de resultados

Algoritmo GRASP

Problema	n	K	m	*Iter*	$\|LRC\|$	z	S	*CPU*
*max_div*_15_3	15	3	2	10	2			
*max_div*_15_3	15	3	2	10	3			
*max_div*_15_3	15	3	2	20	2			
*max_div*_15_3	15	3	2	20	3			
*max_div*_15_3	15	3	3	10	2			
*max_div*_15_3	15	3	3	10	3			
*max_div*_15_3	15	3	3	20	2			
*max_div*_15_3	15	3	3	20	3			
*max_div*_15_3	15	3	4	10	2			
*max_div*_15_3	15	3	4	10	3			
*max_div*_15_3	15	3	4	20	2			
*max_div*_15_3	15	3	4	20	3			
*max_div*_15_3	15	3	5	10	2			
*max_div*_15_3	15	3	5	10	3			
*max_div*_15_3	15	3	5	20	2			
*max_div*_15_3	15	3	5	20	3			
*max_div*_20_3	20	3	2	10	2			
*max_div*_20_3	20	3	2	10	3			
*max_div*_20_3	20	3	2	20	2			
*max_div*_20_3	20	3	2	20	3			
*max_div*_20_3	20	3	3	10	2			
*max_div*_20_3	20	3	3	10	3			
*max_div*_20_3	20	3	3	20	2			
*max_div*_20_3	20	3	3	20	3			
*max_div*_20_3	20	3	4	10	2			
*max_div*_20_3	20	3	4	10	3			
*max_div*_20_3	20	3	4	20	2			
*max_div*_20_3	20	3	4	20	3			
*max_div*_20_3	20	3	5	10	2			
*max_div*_20_3	20	3	5	10	3			
*max_div*_20_3	20	3	5	20	2			
*max_div*_20_3	20	3	5	20	3			
*max_div*_30_3	20	3	2	10	2			
*max_div*_30_3	20	3	2	10	3			
*max_div*_30_3	20	3	2	20	2			
*max_div*_30_3	20	3	2	20	3			
*max_div*_30_3	20	3	3	10	2			
*max_div*_30_3	20	3	3	10	3			
*max_div*_30_3	20	3	3	20	2			
*max_div*_30_3	20	3	3	20	3			
*max_div*_30_3	20	3	4	10	2			
*max_div*_30_3	20	3	4	10	3			
*max_div*_30_3	20	3	4	20	2			
*max_div*_30_3	20	3	4	20	3			
*max_div*_30_3	20	3	5	10	2			
*max_div*_30_3	20	3	5	10	3			
*max_div*_30_3	20	3	5	20	2			
*max_div*_30_3	20	3	5	20	3			

Tabla 3: Tabla de resultados

Algoritmo de ramificación y poda

Problema	n	K	m	z	S	CPU	nodos generados
*max_div*_15_2	15	2	2				
*max_div*_15_2	15	2	3				
*max_div*_15_2	15	2	4				
*max_div*_15_2	15	2	5				
*max_div*_20_2	20	2	2				
*max_div*_20_2	20	2	3				
*max_div*_20_2	20	2	4				
*max_div*_20_2	20	2	5				
*max_div*_30_2	30	2	2				
*max_div*_30_2	30	2	3				
*max_div*_30_2	30	2	4				
*max_div*_30_2	30	2	5				
*max_div*_15_3	15	3	2				
*max_div*_15_3	15	3	3				
*max_div*_15_3	15	3	4				
*max_div*_15_3	15	3	5				
*max_div*_20_3	20	3	2				
*max_div*_20_3	20	3	3				
*max_div*_20_3	20	3	4				
*max_div*_20_3	20	3	5				
*max_div*_30_3	30	3	2				
*max_div*_30_3	30	3	3				
*max_div*_30_3	30	3	4				
*max_div*_30_3	30	3	5				

Tabla 4: Tabla de resultados