Ejercicios Tema 2

Algoritmia para problemas difíciles

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura – Universidad de Zaragoza

6 de noviembre de 2013

Ejercicios sobre algoritmos aproximados

Ejercicio 1 Supón que actúas como consultor para la autoridad portuaria de un pequeño país del Pacífico. En la actualidad su facturación es de varios miles de millones de euros por año, y sus beneficios están limitados casi enteramente por la velocidad a la que pueden descargar los barcos que llegan a puerto.
Aquí está el problema básico que afrontan. Llega un barco, con n contenedores de peso $w_1, w_2, \ldots, w_n$. En el puerto hay esperando un conjunto de camiones, cada uno de los cuales puede transportar K unidades de peso. (Puedes suponer que K y cada w_i son enteros.) Puedes cargar varios contenedores en cada camión, sujeto a la restricción de peso K; el objetivo es minimizar el número de camiones necesarios para transportar todos los contenedores. Este problema es NP-difícil (no tienes que demostrarlo).
Un algoritmo voraz que puedes usar para este problema es el siguiente: Empieza con un camión vacío y empieza a apilar contenedores $1, 2, 3, \ldots$ en él hasta que llegues a un contenedor que haga sobrepasar el límite de peso. Ahora declara este camión "cargado" y envíalo; entonces continúa el proceso con un nuevo camión. Este algoritmo, considerando los camiones de uno en uno, puede no alcanzar la forma más eficiente de cargar todos los contenedores en la colección de camiones disponible.

1. Da un ejemplo de un conjunto de pesos y un valor de K para los que este algoritmo no usa el mínimo número posible de camiones.
2. Demuestra que sin embargo el algoritmo es una 2-aproximación del óptimo.

Ejercicio 2 Supongamos que te dan un conjunto de enteros positivos $A = \{a_1, \ldots, a_n\}$ y un entero positivo B. Un subconjunto $S \subseteq A$ se llama *factible* si la suma de los números de S no sobrepasa B:

$$\sum_{a_i \in S} a_i \leq B.$$

La suma de los números de S se llamará la *suma total* de S.
Te gustaría seleccionar un subconjunto factible S de A cuya suma total sea lo mayor posible.
Ejemplo. Si $A = \{8, 2, 4\}$ y $B = 11$, entonces la solución óptima es $S = \{8, 2\}$.

1. Este es un algoritmo para este problema.

```
1  S = ∅
2  T = 0
3  for i = 1, 2, ..., n
4        if T + a_i ≤ B
5        Añadir a_i a S
6        T = T + a_i
7  Resultado S
```

   Dar una entrada para la que la suma total del conjunto S devuelto por este algoritmo es menos de la mitad de la suma total de cualquier otro posible subconjunto factible de A.

2. Dar un algoritmo de aproximación para este problema basado en el visto en clase para Mochila.

3. Dar un algoritmo de aproximación en tiempo polinómico para este problema con la siguiente garantía: Devuelve un conjunto factible $S \subseteq A$ cuya suma total es de al menos de la mitad del tamaño de la máxima suma total de cualquier conjunto factible $S' \subseteq A$. Tu algoritmo debe tener un tiempo de ejecución como mucho $O(n \log n)$.

Ejercicio 3 Recordemos que en el problema de la mochila tenemos n objetos, cada uno con un peso w_i y un valor v_i. También tenemos un límite de peso W, y el problema es seleccionar un conjunto de elementos S del mayor valor posible sujeto a la condición de que el peso total no exceda W – es decir, $\sum_{i \in S} w_i \leq W$. He aquí una manera mirar el algoritmo de aproximación que hemos diseñado en este capítulo. Si se nos dice que existe un subconjunto O cuyo peso total es $\sum_{i \in O} w_i \leq W$ y cuyo valor total es $\sum_{i \in O} v_i = V$ para algún V, entonces nuestro algoritmo aproximado puede encontrar un conjunto A con peso total $\sum_{i \in A} w_i \leq W$ y valor total al menos $\sum_{i \in A} v_i \geq V/(1+\epsilon)$. Así, el algoritmo aproxima el mejor valor, manteniendo los pesos estrictamente por debajo de W. (Por supuesto, devolver el conjunto O siempre es una solución válida, pero ya que el problema es NP-difícil, no esperamos poder encontrar siempre O; el ratio de aproximación de $1+\epsilon$ significa que los otros conjuntos A, con un poco menos de valor, pueden ser respuestas válidas también.)
Ahora, como es bien sabido, siempre se puede meter un poco más simplemente "sentándose encima de la maleta", es decir, sobrepasando un poco el límite de peso permitido. Esto también sugiere otra manera diferente de formalizar la aproximación para el problema de la mochila, que es la siguiente:
Supongamos , como antes, que nos dan n objetos con pesos y valores, además de los parámetros W y V; y te dicen que hay un subconjunto O cuyo peso total es $\sum_{i \in O} w_i \leq W$ y cuyo valor total es $\sum_{i \in O} v_i = V$. Para un determinado $\epsilon > 0$ fijo, diseña un algoritmo en tiempo polinómico que encuentre un subconjunto de objetos A tal que $\sum_{i \in A} w_i \leq (1+\epsilon)W$ y $\sum_{i \in A} v_i \geq V$. En otras palabras, quieres que A alcance un valor de al menos la cota dada V, pero se le permite exceder el límite de peso W por un factor de $1+\epsilon$.
Ejemplo. Supongamos que te dan cuatro objetos, con pesos y valores como sigue:

$$(w_1, v_1) = (5, 3), \quad (w_2, v_2) = (4, 6)$$
$$(w_3, v_3) = (1, 4), \quad (w_4, v_4) = (6, 11)$$

También te dan $W = 10$ y $V = 13$ (ya que, de hecho, el subconjunto formado por los tres primeros objetos tiene peso total como mucho de 10 y tiene un valor de 13). Por último, te dan $\epsilon = 1$. Esto quiere decir que debes encontrar (con tu algoritmo aproximado) un subconjunto de peso como mucho $(1 + ,1) * 10 = 11$ y valor de al menos 13. Una solución válida sería el subconjunto formado por el primer y cuarto objetos, con valor $14 \geq 13$. (Notar que este es un caso en que puedes alcanzar un valor estrictamente mayor que V ya que se permite sobrecargar un poco la mochila.)

Ejercicio 4 En el problema MAXSAT nos dan un conjunto de cláusulas, y queremos encontrar una asignación que satisface el mayor número posible de ellas. (Nota: Las definiciones de cláusula y literal pueden encontrarse en el guión de la práctica 1.)

1. Demostrar que si este problema se puede resolver en tiempo polinómico, también se puede resolver en tiempo polinómico SAT.

2. Este es un algoritmo muy inocente,

```
1  for cada variable
2        asigna su valor a cierto o falso lanzando una moneda
```

   Supón que tu entrada tiene m cláusulas, de las cuales la número j tiene k_j literales. Demostrar que el número *medio* de cláusulas satisfechas por este algoritmo es

$$\sum_{j=1}^{m} \left(1 - \frac{1}{2^{k_j}}\right) \geq \frac{m}{2}.$$

En otras palabras, es una 2-aproximación en media. Y si las cláusulas contienen todas k literales, entonces el ratio de aproximación mejora a $1 + 1/(2^k - 1)$.

3. ¿Puedes hacer este algoritmo determinista? (*Pista:* En lugar de lanzar una moneda para cada variable, selecciona el valor que satisface más cláusulas de las todavía no satisfechas. ¿Qué fracción de las cláusulas se satisface al final?)

Ejercicio 5 En el problema de MAXCUT se nos da un grafo $G = (V, E)$ con un peso $w(e)$ en cada arista, y queremos separar los vértices en dos conjuntos S y $V - S$ de forma que el peso total de las aristas entre los dos conjuntos sea tan *grande* como sea posible.
Para cada $S \subseteq V$ llamamos $w(S)$ a la suma de todos los $w(e)$ en todas las aristas $e = (u, v)$ tales que $|S \cap \{u, v\}| = 1$. Obviamente, MAXCUT trata de maximizar $w(S)$ sobre todos los subconjuntos de V.
Consideremos el siguiente algoritmo de búsqueda local para MAX CUT:

```
1  comenzar con cualquier S ⊆ V
2  while hay un subconjunto S' ⊆ V tal que |S' − S| = 1 y w(S') > w(S)
3        S = S'
```

1. Demostrar que se trata de una 2-aproximación para MAXCUT.
2. Pero, ¿es un algoritmo en tiempo polinómico?

Ejercicio 6 BINPACKING. Supongamos que se nos da un conjunto de n objetos, donde el tamaño s_i del objeto número i satisface $0 < s_i < 1$. Queremos meter todos los objetos en el número mínimo de cajas tamaño uno. Cada caja puede contener cualquier subconjunto de los objetos cuyo tamaño total no sobrepase 1. La heurística del *primer ajuste* toma cada vez un objeto y lo coloca en la primera caja en la que aun cabe. Sea $S = \sum_{i=1}^{n} s_i$.

1. Argumentar que el número óptimo de cajas necesarias es al menos $\lceil S \rceil$.
2. Argumentar que la heurística del primer ajuste deja a lo sumo una caja llena a menos de la mitad.
3. Demostrar que el número de cajas utilizadas por la heurística de primer ajuste nunca es más que $\lceil 2S \rceil$.
4. Demuestra que la heurística del primer ajuste es una 2-aproximación.
5. Dar una implementación eficiente de la heurística primer ajuste, y analizar su complejidad en tiempo.

Ejercicio 7 Encontrar una implementación del algoritmo APROXSC visto en clase (página 42) con coste en tiempo $O(\sum_{i=1}^{m} |S_i|)$.

Ejercicio 8 Considerar la siguiente heurística para VC (Cobertura de vértices óptima). Construir un árbol recorrido en profundidad del grafo y borrar todas las hojas de este árbol. Demostrar que el tamaño de este cubrimiento es como mucho dos veces mayor que el óptimo.

En caso de entregar alguno de estos ejercicios, la fecha límite es el lunes 2 de diciembre.

Antes de realizar cualquiera de estos ejercicios el alumno debe enviar un correo a `elvira@unizar.es` indicando qué ejercicio desea realizar.

Cualquier fuente utilizada en la resolución de estos ejercicios debe ser indicada claramente en la solución.