CC4102 - Diseño y Análisis de Algoritmos
Auxiliar 10

Prof. Gonzalo Navarro; Aux. Mauricio Quezada

5 de Enero de 2012 :(

1 Ecuaciones binarias

Suponga que tiene un conjunto de n variables binarias $x_1, \ldots, x_n$ y un conjunto de k ecuaciones, donde la ecuación r-ésima es de la forma

$$(x_i + x_j) \bmod 2 = b_r$$

Para dos variables distintas x_i, x_j y algún valor b_r. Considere el problema de encontrar una asignación de valores que maximice el número de ecuaciones que se cumplen.

1. Sea c^* el máximo número de ecuaciones que se cumplen dada una asignación de valores a las variables. Diseñe un algoritmo que produzca una asignación que satisfaga al menos a la mitad de las ecuaciones.

2. Ahora considere el mismo problema pero para una cantidad arbitraria de variables por ecuación.

1.1 Solución

1.1.1 Parte I

Usando la misma estrategia para MAX-3SAT, asignando un 1 o 0 con la misma probabiliadad a cada variable. Sea $E[X_r]$ el valor esperado de la variable que toma el valor 1 si la ecuación r-ésima es satisfecha. De las cuatro posibles valuaciones de la ecuación i, hay dos que causan que se evalúe a 0 ($x_i = x_j = 0$, y $x_i = x_j = 1$), y dos que hacen que evalúe a 1. Por lo tanto, $E[X_r] = 2/4 = 1/2$.

Por linealidad de la esperanza, $E[X] = \sum_r E[X_r] = k/2$. Como el máximo de ecuaciones satisfacibles c^* es a lo más k, se satisfacen $c^*/2$ ecuaciones en promedio.

1.1.2 Parte II

Queremos hacer, igual que para la parte anterior, que $E[X_r] = 1/2$, incluso si hay una cantidad arbiraria de variables en la ecuación r-ésima; en otras palabras, la probabilidad de que la ecuación tome el valor correcto módulo 2 sea exactamente 1/2.

Hay una forma fácil de hacerlo: se asignan valores arbitrarios a todas las variables excepto a la última. Como una asignación lleva a la solución esperada y la otra no, al asignar un valor al azar, la probabilidad de satisfacer a la ecuación r es 1/2.

Otra forma es contar las valuaciones para las cuales una ecuación toma el valor 1, y el valor 0, y verificar que son iguales (por lo que la probabilidad es la misma, 1/2).

Suponga que la ecuación r tiene t términos, por lo que hay 2^t asignaciones posibles. Queremos decir que 2^{t-1} producen una suma 1, y 2^{t-1} producen una suma 0, lo cuál mostrará que $E[X_r] = 1/2$. Probaremos esto por inducción en t.

- $t = 1$; hay dos asignaciones, para $t = 2$ ya lo vimos en la parte anterior.
- Supongamos que la proposición se cumple para $t - 1$, entonces hay exactamente 2^{t-1} formas de tener una suma 0 con t variables, como sigue:
  - 2^{t-2} formas de tener una suma par (0) en las primeras $t - 1$ variables (por inducción), seguido de una asignación 0 al término t-ésimo,
  - 2^{t-2} formas de tener una suma impar (1) en las primeras $t - 1$ variables (por inducción), seguido de una asignación 1 al término t-ésimo.
- Para obtener una suma impar (1), el análisis es análogo al anterior, por lo que la mitad de todas las posibles asignaciónes de valores a x_i producen una suma par, y la mitad una suma impar.

Una vez que tenemos que $E[X_r] = 1/2$, concluimos igual que en la primera parte.

2 Vertex Cover

Un *Vertex Cover* de un grafo no dirigido $G = (V, E)$ es un subconjunto $V' \subseteq V$ tal que si (u, v) es una arista de G, entonces $u \in V'$, o $v \in V'$ (o ambos). El tamaño del Vertex Cover es la cantidad de vértices en él. El *Problema del Vertex Cover en G* es el de encontrar un Vertex Cover de tamaño mínimo de G.

Muestre que, dado $G = (V, E)$, el algoritmo de escoger una arista (u, v) arbitraria de E (y agregarla al Vertex Cover) y remover todas las aristas adyacentes a u y v, hasta recorrer todo el conjunto de aristas, es 2-aproximado.

2.1 Solución

Este algoritmo corre en tiempo $O(|V| + |E|)$ (¿por qué?). Además, el algoritmo es correcto, pues el algoritmo itera por sobre cada arista en E hasta que cada una sea cubierta por algún vértice en el conjunto retornado.

Para mostrar que el algoritmo retorna un vertex cover de a lo más el doble de tamaño que un covering óptimo, sea A el conjunto de aristas que usa nuestro algoritmo. Para cubrir las aristas en A, cualquier vertex cover (en particular, el óptimo C^*), debe incluir al menos un extremo de cada arista en A. No hay dos aristas en A que compartan el mismo extremo, pues una vez que una arista se escoja por el algoritmo, todas las otras que son incidentes a sus extremos son removidas de E. Por lo tanto, no hay dos aristas en A que sean cubiertas por el mismo vértice en C^*, y tenemos la siguiente cota inferior,

$$|C^*| \geq |A|$$

sobre el tamaño del vertex cover óptimo. Cada vez el algoritmo toma una arista para la cual ninguno de sus extremos está en C (el vertex cover retornado), dando una cota superior exacta en el tamaño del vertex cover entregado:

$$|C| = 2|A| \leq 2|C^*|$$

3 Vendedor viajero

Sea $G = (V, E)$ un grafo completo no dirigido, y c una función de costos sobre E tal que (V, c) define un *espacio métrico* (recuerde de la definición de k-servers). Muestre un algoritmo 2-aproximado para el problema del vendedor viajero sobre G con costos dados por c.

3.1 Solución

El algoritmo usará el *árbol cobertor mínimo* (MST) para generar de este un tour. El algoritmo es como sigue:

- Approx-TSP-Tour(G, c)
  1. Seleccione un vértice $r \in V$
  2. Calcule el MST T de G con raíz r
  3. Sea H una lista de vértices de T, de acuerdo a visitar el árbol en preorden
  4. **retornar** el ciclo hamiltoniano generado de T uniendo el primer y último vértice

El árbol cobertor puede ser calculado en $O(V + E)$ o bien en $O(E \log V)$ (como vimos con Union-Find). En cualquier caso, el tiempo de Approx-TSP-Tour es polinomial en n (la cantidad de nodos).

Sea H^* un tour óptimo para el grafo. Obtenemos un MST sacando cualquier arista del tour, y cada costo es no-negativo. Por lo tanto, el peso del MST T dado por nuestro algoritmo da una cota inferior en el costo del tour óptimo: $c(T) \leq c(H^*)$

Una *caminata completa* (*full walk*) de T lista los vértices cuando son visitados del árbol y además cuando se devuelve a la raíz para vistiar otro subárbol. Llamemos a la caminata de T como W; en el ejemplo visto en clases, la caminata completa da la lista:

$$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$$

Como la caminata pasa por cada arista de T 2 veces, tenemos que $c(W) = 2c(T)$ (abusando de la notación), por lo que $c(W) \leq 2c(H^*)$. Generalmente W no es un tour (ya que puede visitar un nodo más de una vez), sin embargo, usando la desigualdad triangular podemos quitar nodos visitados y el costo no aumenta (si eliminamos un nodo v de W entre la visita de u a w, el orden resultante especifica ir directamente de u a w). Usando esto, podemos quitar todos los nodos excepto la primera visita a cada uno de W, por ejemplo

$$a, b, c, h, d, e, f, g$$

Equivalentemente, este orden equivale a visitar T en preorden. Sea H el ciclo correspondiente a esta visita. Es un ciclo hamiltoniano, pues cada vértice es visitado exactamente una vez, y de hecho

es el ciclo calculado por Approx-TSP-Tour. Como H se obtiene eliminando vértices de W, tenemos que $c(H) \leq c(W)$ y por lo tanto $c(H) \leq 2c(H^*)$ lo cual demuestra el resultado.

4 Bonus: Bin Packing

Suponga que tiene n objetos de volúmenes $0 < s_i \leq 1$, $i \in \{1, \ldots, n\}$ y una cantidad arbitraria de cajas de volumen $V = 1$. El objetivo es minimizar la cantidad de cajas utilizadas para guardar todos los objetos. Muestre una 2-aproximación para este problema.

4.1 Solución

Llamemos a esta estrategia "First-Fit": *Si el objeto actual no cabe en ninguna caja de las ya ocupadas, utilizar una nueva caja.*

Es claro ver que el algoritmo es correcto y corre en tiempo polinomial.

Para mostrar que es 2-aproximado, suponga que FF usa m cajas. Podemos afirmar que al menos $m-1$ cajas están a más de la mitad de su capacidad, en caso contrario, hay al menos dos cajas con menos de la mitad de su capacidad y por lo tanto podemos juntarlas hasta tener una sola caja con más de la mitad de su capacidad.

Por otra parte, $\sum_{i=1}^{n} s_i \leq C^*$, donde C^* es la cantidad de cajas óptima (pues las cajas tienen volumen unitario), pero $\sum_{i=1}^{n} s_i > \frac{m-1}{2}$ por lo anterior, y entonces $2C^* > m - 1$ por lo que $m < 2C^* + 1$, y la cantidad de cajas utilizadas por FF es a lo más el doble que las del algoritmo óptimo.