UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación
e Inteligencia Artificial

Modelos Avanzados de Computación

Práctica 3

Reducciones

Curso 2014-2015

Doble Grado en Ingeniería Informática y Matemáticas

Práctica 3

Reducciones

Las reducciones se utilizan, en complejidad computacional, para mostrar que un problema B es, al menos, tan difícil como un problema A mostrando un algoritmo que podría utilizarse para resolver A dada una subrutina que resuelve B.

Una reducción $A \leq B$ muestra que el problema A es, como mucho, tan difícil como el problema B, además de mostrar que B es al menos tan difícil como A.. Por tanto, la reducción nos da una cota inferior de la complejidad de B y una cota superior de la complejidad de A.

El término puede resultar confuso, ya que decir que A se puede reducir a B no quiere decir que B sea más sencillo o pequeño que A, sino que A puede verse como un caso particular de B. Pero B es más general que A y, por tanto, puede ser más difícil ("B es lo suficientemente expresivo para describir los objetivos y las restricciones de A").

1. **El problema de la parada** (una vez más): Definimos la función $HALT(p, x)$ como una función booleana que nos indica si la máquina de Turing M_p representada por p para en un número finito de pasos para la entrada x. Por otro lado, sabemos (por diagonalzación) que existe una función no computable UC definida como se describe a continuación. La función $UC(s)$ es 0 si la máquina de Turing codificada por s devuelve 1 cuando recibe s como entrada; esto es, $UC(s) = 0$ cuando $M_s(s) = 1$. En cualquier otro caso, cuando la máquina de Turing devuelve cualquier otro valor o, simplemente, no para, $UC(s) = 1$. Dada la existencia de la función no computable UC, demostrar por reducción que la función $HALT$ tampoco es computable.

   > NOTA: Para demostrar que UC no es computable, basta con que supongamos que UC es computable. Entonces, existiría una máquina de Turing M tal que $M(s) = UC(s)$ para cualquier cadena de bits s. En particular, debería verificarse que $M(M) = UC(M)$, lo cual, por la propia definición de UC, es imposible.

   [Arora & Barak: "Computational Complexity", 1.5.1, pp. 22-23]

2. **Caminos disjuntos** [Disjoint Paths]: Dos caminos en un grafo se dicen disjuntos [edge-disjunct] si no comparten aristas en común. Sin embargo, sí permitimos que compartan vértices. Demuestre cómo resolver el siguiente problema de decisión a partir de un algoritmo que nos permita calcular el flujo máximo en una red [Max Flow]: Dados un grafo dirigido (G), dos vértices (s, t) y un número entero (k), determinar si existen k caminos disjuntos de s a t.

   [Moore & Mertens: "The Nature of Computation", problem 3.48, pp. 88]

   [Kleinberg & Tardos: "Algorithm Design", Section 7.6, pp. 373-378]

3. **Spin glass** (`http://en.wikipedia.org/wiki/Spin_glass`): Un "vidrio de espín" es un sistema magnético en el que el acoplamiento entre los momentos magnéticos de los distintos átomos es aleatorio (algo así como un imán desordenado). Formalmente, puede verse como un grafo en el que el vértice i-ésimo tiene un espín $s_i = \pm 1$ que representa si el campo magnético del átomo i-ésimo apunta hacia arriba o hacia abajo. Cada arista (i, j) del grafo tiene asociada una fuerza de interacción J_{ij} que indica hasta qué punto interactúan s_i y s_j. Además, cada vértice i está sometido a un campo externo h_i. Dada una configuración del sistema (un conjunto de valores para los s_i), su energía es

   $$E(\{s_i\}) = -\sum_{ij} J_{ij} s_i s_j - \sum_i h_i s_i$$

   La arista (i, j) es ferromagnética si $J_{ij} > 0$ y antiferromagnética si si $J_{ij} < 0$; esto es, el ferromagnetismo tiende a alinear los espines. El campo externo es positivo o negativo en función de si queremos que s_i sea $+1$ o -1, respectivamente.

   Un sistema como el descrito tiende a minimizar su energía, al menos a temperatura cero. Muestre cómo calcular el estado de mínima energía del sistema ('ground state' para los físicos) para materiales ferromagnéticos ($J_{ij} \geq 0$) reduciendo el problema al de calcular un corte mínimo en un grafo [Min Cut], problema que se puede resolver en tiempo polinómico.

   [Moore & Mertens: "The Nature of Computation", problem 3.50, pp. 88-89]

4. **Problemas equivalentes:** Dos problemas A y B son equivalentes cuando A se puede reducir a B y B también se puede reducir a A. Muestre la equivalencia entre los elementos de los siguientes conjuntos de problemas:

   2 problemas definidos sobre una lista de enteros:

   - Integer Partitioning: Dada una lista de enteros positivos $S = \{x_1..x_k\}$, ¿existe una partición balanceada? Esto es, un subconjunto $A \subseteq \{1..k\}$ tal que $\sum_{i \in A} x_i = \sum_{i \notin A} x_i$.

- Subset Sum: Dada una lista de enteros positivos $S = \{x_1..x_k\}$ y un entero t, ¿existe un subconjunto de S tal que sus elementos sumen t? Esto es, un subconjunto $A \subseteq \{1..k\}$ tal que $\sum_{i \in A} x_i = t$.

[Moore & Mertens: "The Nature of Computation", 4.2.3, pp. 105-107 & exercise 4.13, p. 122]

3 problemas definidos sobre un grafo $G = (V, E)$:

- Independent Set: Dado un grafo G y un entero k, ¿tiene G un conjunto independiente de tamaño k o mayor? Un conjunto de vértices $S \subseteq V$ es independiente si no incluye dos vértices adyacentes.
- Vertex Cover: Dado un grafo G y un entero k, ¿tiene G una cobertura de tamaño k o menor? Un conjunto de vértices $S \subseteq V$ es una cobertura del grafo G si, para toda arista $e \in E$, al menos uno de los extremos de e está en S.
- Clique: Dado un grafo G y un entero k, ¿tiene G un clique de tamaño k o mayor? Un clique es un subgrafo completo: un conjunto de vértices $S \subseteq V$ forma un clique si todos sus vértices son adyacentes.

[Moore & Mertens: "The Nature of Computation", 4.2.4, pp. 107-109]

Otras reducciones relacionadas

Bipartite Matching $\leq$ Max Flow

Kleinberg & Tardos: "Algorithm Design", 7.5, pp. 367-373

Moore & Mertens: "The Nature of Computation", 3.8, pp. 74-75

Circuit Satisfiability $\leq$ 3-SAT

Kleinberg & Tardos: "Algorithm Design", 8.4, pp. 471-472

Moore & Mertens: "The Nature of Computation", 5.2.2, pp. 131-133

3-Dimensional Matching $\leq$ Subset Sum

Kleinberg & Tardos: "Algorithm Design", 8.8, pp. 491-493

Hamiltonian Cycle $\leq$ Traveling Salesman

Kleinberg & Tardos: "Algorithm Design", 8.5, p. 479

Independent Set $\leq$ Set Packing

Kleinberg & Tardos: "Algorithm Design", 8.1, pp. 458-459

Graph 2-Coloring $\leq$ 2-SAT

Moore & Mertens: "The Nature of Computation", exercise 4.8, p. 105

Graph 3-Coloring $\leq$ SAT

Moore & Mertens: "The Nature of Computation", exercise 4.5, p. 102

Graph 3-Coloring $\leq$ Planar Graph 3-Coloring

Moore & Mertens: "The Nature of Computation", 4.2.1, pp. 99-100

Graph k-Coloring $\leq$ Graph (k+1)-Coloring

Moore & Mertens: "The Nature of Computation", exercise 4.3, p. 100

Max Flow $\leq$ Min Cut

Kleinberg & Tardos: "Algorithm Design", 7.2, pp. 346-350

Moore & Mertens: "The Nature of Computation", 3.7, pp. 71-73

Min Cut $\leq$ Max Flow

Kleinberg & Tardos: "Algorithm Design", 7.2, pp. 346-350

Moore & Mertens: "The Nature of Computation", 3.7, pp. 71-73

2-SAT $\leq$ Reachability

Moore & Mertens: "The Nature of Computation", 4.2.2, pp. 102-104

3-SAT $\leq$ 3-Dimensional Matching

Kleinberg & Tardos: "Algorithm Design", 8.6, pp. 481-485

3-SAT $\leq$ Graph 3-Coloring

Kleinberg & Tardos: "Algorithm Design", 8.7, pp. 487-490

3-SAT $\leq$ Hamiltonian Cycle

Kleinberg & Tardos: "Algorithm Design", 8.5, pp. 475-479

3-SAT $\leq$ Independent Set

Kleinberg & Tardos: "Algorithm Design", 8.3, pp. 460-462

k-SAT $\leq$ 3-SAT

Moore & Mertens: "The Nature of Computation", 4.2.2, pp. 104-105

Vertex Cover $\leq$ Set Cover

Kleinberg & Tardos: "Algorithm Design", 8.1, pp. 456-458