Problemas, algoritmos y optimización

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Área de Optimización Combinatoria
Departamento de Sistemas

3 a 7 de septiembre de 2012

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Contenido

Resolviendo problemas matemáticos

George Pólya (1887-1985)

Dice[a] que se pueden resolver en cuatro pasos:

1. Entiende el problema.
2. Crea un plan.
3. Sigue el plan.
4. Interpreta el resultado.

[a] *How to Solve It, Princeton University Press, 1945*

Entiende el problema

Aunque lo parezca, esto no es obvio:

1. ¿Qué te piden encontrar o demostrar?
2. ¿Puedes explicar el problema?
3. ¿Puedes hacer un diagrama que te ayude?
4. ¿Tienes suficiente información?

Crea un plan

Hay muchas formas razonables de resolver un problema:

1. Adivina y verifica.
2. Construye una lista ordenada.
3. Elimina posibilidades.
4. Usa la simetría.
5. Considera casos especiales.
6. Resuelve una ecuación.

Sigue el plan

Esto suele ser más fácil:

1. Necesitas cuidado y paciencia.
2. Insiste con el plan que elegiste.
3. Si no funciona, cambia el plan.
4. Así resuelven problemas los profesionales.

Interpreta el resultado

Reflexiona sobre lo que aprendiste:

1. ¿Qué sirvió y qué no sirvió?
2. ¿Cómo usar esto para resolver otros problemas?

George Pólya

También dice que si no puedes resolver el problema entonces:

1. Encuentra un problema más sencillo.
2. Encuentra un problema relacionado.

¿Qué es un problema computacional?

De manera muy informal

Un problema $\mathcal{P}$ es una relación (función) entre un conjunto $\mathcal{I}$ de instancias y un conjunto $\mathcal{S}$ de soluciones.

1. Si $(i, s) \in \mathcal{P}$ diremos que s es una solución para la instancia i.
2. Usualmente se define $\mathcal{P}$ por comprensión: se describe el conjunto de instancias y luego se describen las soluciones para cada instancia.

Ejemplos de problemas

1. Por comprensión: $\mathcal{P} = \{((a, b), x) : ax + b = 0 \text{ con } a, b, x \in \mathcal{R}\}$.
2. Por extensión: $\mathcal{P} = \{(3, 1), (4, 1), (5, 9), (2, 6), (5, 4)\}$.

Tipos de problemas

De manera muy general hay:

1. Problemas triviales.
2. Problemas de decisión.
3. Problemas de búsqueda.
4. Problemas de conteo.
5. Problemas de optimización.

Problemas triviales

Parece trivial

En un problema trivial la solución es la misma para todas las instancias.
Lo interesante es saber si un problema es trivial.

Ejemplos

1. Problema de Collatz.
2. Conjetura de Goldbach.
3. Teorema de los cuatro colores.
4. Último teorema de Fermat.

Problema de Collatz

Función collatz(n)

1. Si $n = 1$ termina.
2. Si n es par entonces llama a collatz($n/2$).
3. Llama a collatz($3n + 1$).

Enunciado

¿Terminará el proceso anterior para un valor dado de n? (Erdös 500 US)

$$10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1.$$

¿Qué se sabe?

Hasta 2012 se sabe[a] que termina para toda $n \leq 5 \times 2^{60}$.

[a] `http://www.ieeta.pt/~tos/3x+1.html`

Conjetura de Goldbach

Enunciado

Sea $n \geq 4$ un entero positivo par. ¿Se puede escribir n como suma de dos primos? (Faber 1 000 000 US)

$$4 = 2 + 2, 6 = 3 + 3, 8 = 5 + 3, 10 = 5 + 5, 12 = 7 + 5, \ldots$$

¿Qué se sabe?

Hasta 2012 se sabe[a] que es cierta para $n \leq 5 \times 2^{18}$.

[a] `http://www.ieeta.pt/~tos/goldbach.html`

Teorema de los cuatro colores

Enunciado

Considere un mapa plano dividido en regiones conexas. Se desea colorear el mapa de modo que cada región reciba un color distinto de sus regiones vecinas. ¿Se podrá colorear siempre con cuatro o menos colores?

Demostraciones

Appel y Haken demostraron el teorema en 1976[a]. Ahora se conocen dos nuevas demostraciones debidas a Robertson, Sanders, Seymour y Thomas (1996[b] y 2001). Todas ellas usan computadoras.

---

[a] *Every Planar Map is Four Colorable*, Illinois Journal of Mathematics, 1977.
[b] *The Four-Colour Theorem*, J. Combin. Theory Ser. B, 1997.

Último teorema de Fermat

Enunciado

Sea $n \geq 3$ un entero. ¿Existirán soluciones enteras $x, y, z > 0$ de la ecuación $x^n + y^n = z^n$?

Demostración

Muchas personas trabajaron en este problema, pero la última pieza de la demostración se debe a Andrew Wiles en 1995[a].

[a] *Modular elliptic curves and Fermat's Last Theorem*, Annals of Mathematics, 1995.

Problemas de decisión

Soluciones binarias

En un problema de decisión las instancias se dividen en dos clases: *sí* y *no*.

Observación

Cualquier problema $\mathcal{P}$ con conjunto de instancias $\mathcal{I}$ y de soluciones $\mathcal{S}$ tiene un problema de decisión asociado $\mathcal{D}$ con el mismo conjunto de instancias y $\mathcal{D}(i) =$ sí exactamente cuando existe $s \in \mathcal{S}$ tal que $(i, s) \in \mathcal{P}$.

Equivalencia

De manera equivalente, un problema de decisión es un subconjunto $\mathcal{D}$ de un conjunto de instancias $\mathcal{I}$ (las instancias con solución *sí*).

Problemas de decisión

Ejemplos

1. ¿Será par el entero n?
2. ¿Será impar el entero n?
3. ¿Será primo el entero n?
4. ¿Será compuesto el entero n?
5. ¿Terminará el proceso de Collatz si comenzamos con el entero n?
6. ¿Se podrá escribir el entero n como suma de dos primos?
7. ¿Se podrá colorear un mapa dado con cuatro colores o menos?
8. ¿Tendrá soluciones enteras un sistema de ecuaciones polinomiales?
9. ¿Terminará la ejecución de un programa dado?

Problema de satisfacibilidad

Satisfacibilidad

Dada una expresión booleana ¿habrá una asignación de valores de verdad a las variables que la hagan verdadera?

$$\neg x_1 \wedge x_2 \vee (x_3 \vee \neg x_4).$$

Sat

Dada una expresión booleana en forma normal conjuntiva:

$$(\neg x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_4).$$

3Sat

Dada una expresión booleana con tres literales por cláusula:

$$(\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4).$$

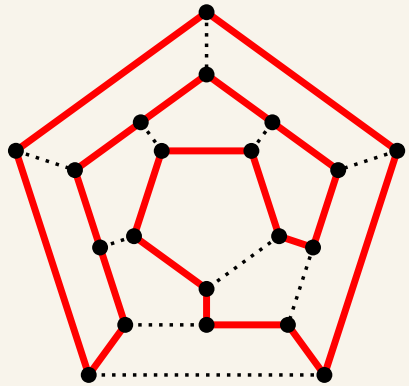
Problema del ciclo hamiltoniano

Ciclo hamiltoniano

Un ciclo que pasa exactamente una vez por cada vértice de una gráfica.

Hamiltonian

Dada una gráfica G ¿tiene un ciclo hamiltoniano?



Problemas de búsqueda

Soluciones generales

En un problema de búsqueda se desea obtener todas las soluciones de cada instancia.

Observación

Cualquier problema $\mathcal{P}$ con conjunto de instancias $\mathcal{I}$ y de soluciones $\mathcal{S}$ tiene un problema de búsqueda asociado $\mathcal{B}$ con el mismo conjunto de instancias y $\mathcal{B}(i) = \{s : (i, s) \in \mathcal{P}\}$.

Problemas de búsqueda

Ejemplos

1. ¿Cuáles son los divisores del entero n?
2. ¿Cuáles son los múltiplos del entero n?
3. ¿Cuáles son los factores primos del entero n?
4. ¿Qué enteros aparecen en el proceso de Collatz si comenzamos con n?
5. ¿Cómo se puede escribir el entero n como suma de dos primos?
6. ¿Como se puede colorear un mapa dado con cuatro colores o menos?
7. ¿Qué soluciones enteras tiene un sistema de ecuaciones polinomiales?

Problema de satisfacibilidad

Satisfacibilidad

Dada una expresión booleana ¿cuáles asignaciones de valores de verdad a las variables la hacen verdadera?

$$\neg x_1 \wedge x_2 \vee (x_3 \vee \neg x_4).$$

Sat

Dada una expresión booleana en forma normal conjuntiva:

$$(\neg x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_4).$$

3Sat

Dada una expresión booleana con tres literales por cláusula:

$$(\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4).$$

Problemas de conteo

Cantidad de soluciones

En un problema de conteo se desea obtener la cantidad de soluciones de cada instancia.

Observación

Cualquier problema $\mathcal{P}$ con conjunto de instancias $\mathcal{I}$ y de soluciones $\mathcal{S}$ tiene un problema de conteo asociado $\mathcal{C}$ con el mismo conjunto de instancias y $\mathcal{C}(i) = |\{s : (i, s) \in \mathcal{P}\}|$.

Problemas de conteo

Ejemplos

1. ¿Cuántos divisores tiene el entero n?
2. ¿Cuántos múltiplos tiene el entero n?
3. ¿Cuántos factores primos tiene el entero n?
4. ¿Cuántos pasos dura el proceso de Collatz si comenzamos con n?
5. ¿De cuántas formas se puede escribir n como suma de dos primos?
6. ¿De cuántas formas se puede colorear un mapa con cuatro colores?
7. ¿Cuántas soluciones tiene un sistema de ecuaciones polinomiales?

Problema de satisfacibilidad

Satisfacibilidad

Dada una expresión booleana ¿cuántas asignaciones de valores de verdad a las variables la hacen verdadera?

$$\neg x_1 \wedge x_2 \vee (x_3 \vee \neg x_4).$$

Sat

Dada una expresión booleana en forma normal conjuntiva:

$$(\neg x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_4).$$

3Sat

Dada una expresión booleana con tres literales por cláusula:

$$(\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4).$$

Problemas de optimización

Soluciones óptimas

En un problema de optimización se desea obtener las mejores soluciones de cada instancia. Cada instancia $i \in \mathcal{I}$ incluye una función objetivo f_i de su conjunto de soluciones $\mathcal{S}_i$ a un conjunto con un orden total (usualmente los reales) y se desea obtener el menor valor posible de $f_i(s)$ con $s \in \mathcal{S}_i$.

Observaciones

A los elementos de $\mathcal{S}_i$ se les llama soluciones factibles para i. A cualquier $s \in \mathcal{S}_i$ que minimice el valor de $f_i(s)$ se le llama solución óptima. No todos los problemas de optimización tienen soluciones óptimas o factibles.

Problema de satisfacibilidad máxima

Max Sat

Dada una expresión booleana en forma normal conjuntiva ¿cuál es la cantidad máxima de cláusulas que se pueden satisfacer simultáneamente?

$$(\neg x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_4).$$

Max 3Sat

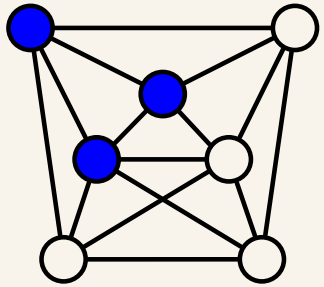
Dada una expresión booleana con tres literales por cláusula:

$$(\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3 \vee \neg x_4).$$

Problema del clan máximo

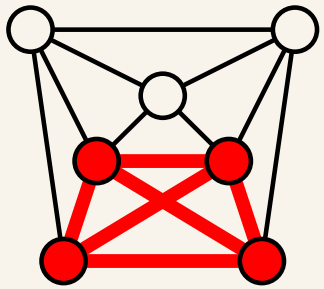
Clan

Un clan K de una gráfica G es una subgráfica completa de G. El tamaño de K es el número de vértices de K.



Max Clique

Dada una gráfica G ¿cuál es el tamaño máximo de sus clanes?



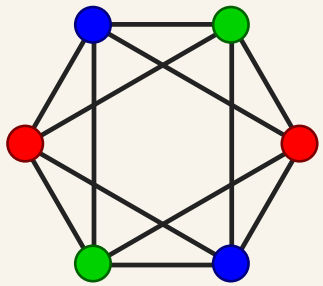
Problema de la coloración mínima

Coloración

Sea $G = (V, E)$ una gráfica. Una función $f : V \to \{1, \ldots, n\}$ es una coloración con n colores de G si $f(u) \neq f(v)$ para toda arista $uv \in E$.

Min Color

Dada una gráfica G ¿cuál es el mínimo número de colores de una coloración de G?



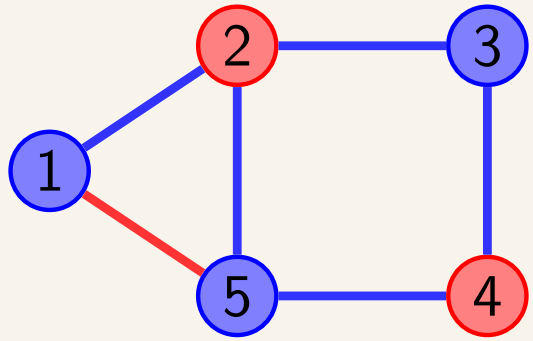
Problema del corte máximo

Corte

Sea $G = (V, E)$ una gráfica y $U \subseteq V$. El corte $C = \delta(U)$ definido por U es el conjunto de las aristas con un vértice en U y el otro en $V \setminus U$.

Max Cut

Dada una gráfica G ¿cuál es el tamaño máximo de sus cortes?



Problema del empaquetamiento de conjuntos

Conjuntos

Sea S un conjunto finito y $\mathcal{F}$ una familia de subconjuntos de S. Una subfamilia $\mathcal{E} \subseteq \mathcal{F}$ es un empaquetamiento de tamaño $|\mathcal{E}|$ si los elementos de $\mathcal{E}$ son disjuntos a pares.

Max Set Packing

¿Cuál es el tamaño máximo de un empaquetamiento de $\mathcal{F}$?

Problema de la cubierta de conjuntos

Conjuntos

Sea S un conjunto finito y $\mathcal{F}$ una familia de subconjuntos de S. Una subfamilia $\mathcal{E} \subseteq \mathcal{F}$ es una cubierta de tamaño $|\mathcal{E}|$ si la unión de los elementos de $\mathcal{E}$ es S.

Min Set Cover

¿Cuál es el tamaño mínimo de una cubierta de $\mathcal{F}$?

Listas de problemas

El artículo de Karp

Reducibility Among Combinatorial Problems, Complexity of Computer Computations, Plenum, 85–103, 1972.

El libro de Garey y Johnson

Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.

El compendio de Crescenzi y Kann

A Compendium of NP Optimization Problems. Disponible en línea en `http://www.nada.kth.se/~viggo/wwwcompendium/`

Contenido

¿Qué es un algoritmo?

De manera muy informal

Un algoritmo es un método efectivo expresado como una lista finita de instrucciones bien definidas. Comenzando desde un estado inicial y con una entrada (posiblemente vacía), las instrucciones describen un proceso determinista que, al ejecutarse, procederá a través de una cantidad finita de estados consecutivos, eventualmente generando una salida (posiblemente vacía) y terminando en un estado final.

Visto como función

Un algoritmo $\mathcal{A}$ es una función de su conjunto de entradas $\mathcal{I}$ a su conjunto de salidas $\mathcal{S}$. Si se ejecuta $\mathcal{A}$ con la entrada $e \in \mathcal{I}$ y al terminar produce la salida $s \in \mathcal{S}$, diremos que $\mathcal{A}(e) = s$.

Definiciones de algoritmo

Cronológicamente

- 1870: Máquina lógica de Jevon.
- 1881: Premisas de Venn.
- 1943: Cálculos de Kleene.
- 1952: Máquinas de Turing.
- 1954: Caracterización de Markov.
- 1963: Caracterización de Gödel.
- 1967: Caracterización de Minsky.
- 1968: Caracterización de Knuth.
- 1972: Caracterización de Stone.
- 1995: Caracterización de Soare.
- 2000: Caracterización de Gurevich.
- 2006: Descripciones de Sipser.

¿Cuándo no es un algoritmo?

Si viola cualquier condición

1. Si la lista de instrucciones no es finita.
2. Si alguna instrucción no está bien definida:
   1. Indefinida.
   2. Ambigua.
   3. No efectiva.
   4. Aleatoria.
   5. No determinista.
3. Estado inicial ambiguo o indefinido.
4. Estado siguiente ambiguo o indefinido.
5. No termina para alguna entrada.

¿Cómo expresamos algoritmos?

Entre otros:

1. Lenguaje natural.
2. Seudocódigo.
3. Diagramas de flujo.
4. Lenguajes de programación.
5. Tablas de control.
6. Máquinas de Turing.

Observación

Todos estos sistemas permiten expresar cosas que no son algoritmos (por ejemplo, procedimientos no deterministas o que nunca terminan).

Algoritmos que resuelven problemas

Diremos que un algoritmo $\mathcal{A} : \mathcal{I} \to \mathcal{S}$ resuelve un problema $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ si $\mathcal{A}(e) = \mathcal{P}(e)$ para toda $e \in \mathcal{I}$.

Observación

En realidad basta que el dominio de $\mathcal{A}$ incluya al dominio de $\mathcal{P}$ y que el contradominio de $\mathcal{A}$ incluya a la imagen de $\mathcal{P}$. Es decir, un algoritmo $\mathcal{A}$ puede resolver un problema más general que $\mathcal{P}$.

Ejemplo de algoritmo

Función euclides(A, B)

1. Mientras $B \neq 0$:
   1. $C \leftarrow B$.
   2. $B \leftarrow A \mod B$.
   3. $A \leftarrow C$.
2. Regresa A.

Máximo común divisor

No es difícil ver que euclides$(A, B) = mcd(A, B)$.

Límites algorítmicos

Indecidibilidad

No todos los problemas tienen algoritmos que los resuelvan:

1. Problemas indecidibles.
2. Funciones no computables.

Máquina de Turing

Definición

Una máquina de Turing $\mathcal{T}$ es una tupla $(Q, \Gamma, b, \Sigma, s, F, \delta)$ donde:

- Q es un conjunto finito de estados.
- Γ es un conjunto finito de símbolos, el alfabeto de cinta.
- $b \in \Gamma$ es el símbolo blanco.
- $\Sigma \subseteq \Gamma \setminus b$ es un conjunto finito de símbolos, el alfabeto de entrada.
- $s \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados de aceptación.
- $\delta : Q \setminus F \times \Gamma \to Q \times \Gamma \times \{L, R\}$ es una función de transición, donde L es el movimiento a la izquierda y R es el movimiento a la derecha.

Problema del detenimiento

Definición

Dada una descripción de una máquina de Turing $\mathcal{T}$ y de su entrada e, determine si $\mathcal{T}(e)$ se detiene o si se ejecuta para siempre.

Teorema

El problema del detenimiento es indecidible, es decir, no existe una máquina de Turing $\mathcal{D}$ que lea como entrada $\mathcal{T}$ y e, que siempre se detenga y que siempre emita la respuesta correcta.

Errores comunes

Dos máquinas

Considere dos máquinas de Turing $\mathcal{S}$ y $\mathcal{N}$ que siempre responden *sí* y *no*, respectivamente. Entonces una de $\mathcal{S}(\mathcal{T}, e)$ y $\mathcal{N}(\mathcal{T}, e)$ es la respuesta correcta. ¿Pero cuál?

Simulación

Considere una máquina de Turing $\mathcal{S}$ que simula la ejecución de $\mathcal{T}(e)$. Entonces $\mathcal{S}(\mathcal{T}, e)$ se detiene si y sólo si $\mathcal{T}(e)$ se detiene. ¿Pero cuándo?

Tesis de Church

Tesis

Toda función calculable se puede realizar en una máquina de Turing[a].

[a] *An Unsolvable Problem of Elementary Number Theory*. American Journal of Mathematics 58, 345–363, 1936.

Tesis física

Toda función calculable con un dispositivo físico se puede realizar en una máquina de Turing.

Alternativas a los algoritmos

Opciones

1. Permitir que no termine siempre.
2. Permitir ejecución no determinista.
3. Permitir instrucciones aleatorias.
4. Permitir que no resuelva el problema.

Algoritmos parciales

Algoritmos parciales y problemas

Un algoritmo parcial $\mathcal{A} : \mathcal{I} \to \mathcal{S}$ para un problema $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ es tal que cada vez que la ejecución de $\mathcal{A}(e)$ termina entonces $\mathcal{A}(e) = \mathcal{P}(e)$.

En otras palabras

Un algoritmo parcial para un problema nunca se equivoca, pero en ocasiones no termina.

Ejemplo

La simulación de un programa para decidir si termina o no.

Algoritmos no deterministas

Definición

Una máquina de Turing no determinista es una tupla $(Q, \Gamma, b, \Sigma, s, F, \delta)$:

- Q es un conjunto finito de estados.
- Γ es un conjunto finito de símbolos, el alfabeto de cinta.
- $b \in \Gamma$ es el símbolo blanco.
- $\Sigma \subseteq \Gamma \setminus b$ es un conjunto finito de símbolos, el alfabeto de entrada.
- $s \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados de aceptación.
- $\delta \subseteq (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R\})$ es una relación de transición, donde L es el movimiento a la izquierda y R a la derecha.

Diferencia

Determinista

Una máquina de Turing $\mathcal{T}$ determinista acepta una entrada e si la ejecución de $\mathcal{T}(e)$ lleva a $\mathcal{T}$ a un estado final $f \in F$.

No determinista

Una máquina de Turing $\mathcal{T}$ no determinista acepta una entrada e si alguna ejecución de $\mathcal{T}(e)$ lleva a $\mathcal{T}$ a un estado final $f \in F$.

Algoritmos Las Vegas

Algoritmos Las Vegas y problemas

Un algoritmo Las Vegas[a] $\mathcal{A} : \mathcal{I} \to \mathcal{S}$ para un problema $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ permite instrucciones aleatorias y es tal que $\mathcal{A}(e) = \mathcal{P}(e)$ para toda $e \in \mathcal{I}$.

[a] L. Babai, *Monte-Carlo algorithms in graph isomorphism testing*, Université de Montréal, D.M.S. 79-10, 1979.

En otras palabras

Un algoritmo Las Vegas para un problema nunca se equivoca, pero las ejecuciones con la misma entrada pueden ser distintas.

Ordenamiento y selección

Problema del ordenamiento

Dada una lista $S = (s_1, s_2, \ldots, s_n)$ de n elementos se desea encontrar una nueva lista $S' = (s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(n)})$ con los mismos elementos pero ordenados ascendentemente.

Problema de selección

Dada una lista $S = (s_1, s_2, \ldots, s_n)$ de n elementos y un entero $1 \leq k \leq n$ se desea encontrar el k-ésimo elemento más pequeño de S.

Ejemplos

1. Quicksort aleatorizado.
2. Selección aleatorizada.

Quicksort aleatorizado

Función quicksort(S)

1. Si $|S| \leq 1$ entonces regresa S.
2. Si no entonces:
   1. Escoge aleatoriamente un elemento a de S (el pivote).
   2. Sean $S_<$, $S_=$ y $S_>$ los elementos de S que son $< a$, $= a$ y $> a$ respectivamente.
   3. Regresa (quicksort($S_<$), $S_=$, quicksort($S_>$)).

Dos ejecuciones

1. $3, 2, 5, 6, 4, 1 \rightarrow 2, 1, 3, 5, 6, 4 \rightarrow \cdots \rightarrow 1, 2, 3, 4, 5, 6.$
2. $3, 2, 5, 6, 4, 1 \rightarrow 3, 2, 1, 4, 5, 6 \rightarrow \cdots \rightarrow 1, 2, 3, 4, 5, 6.$

Selección aleatorizada

Función selección(S, k)

1. Si $|S| \leq 1$ entonces regresa S_1.
2. Si no entonces
   1. Escoge aleatoriamente un elemento a de S (el pivote).
   2. Sean $S_<$, $S_=$ y $S_>$ los elementos de S que son $< a$, $= a$ y $> a$ respectivamente.
   3. Sea $j = |S_<|$.
   4. Si $k = j + 1$ regresa a.
   5. Si no, si $k \leq j$ regresa selección$(S_<, k)$
   6. Si no, regresa selección$(S_>, k - j - 1)$.

Análisis de quicksort y selección

Correctitud

Se puede demostrar por inducción en el número de elementos de S que los dos algoritmos regresan lo esperado.

Tiempo de ejecución

Nos estamos adelantando, pero... quicksort hace entre $n \log n$ y n^2 pasos, mientras que selección hace entre n y n^2 pasos.

Algoritmos Monte Carlo

Algoritmos Monte Carlo y problemas

Un algoritmo Monte Carlo $\mathcal{A} : \mathcal{I} \to \mathcal{S}$ para un problema $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ permite instrucciones aleatorias y es tal que $\mathcal{A}(e) = \mathcal{P}(e)$ para toda $e \in \mathcal{I}$ con probabilidad positiva.

En otras palabras

Un algoritmo Monte Carlo para un problema a veces se equivoca y las ejecuciones con la misma entrada pueden ser distintas.

Ejemplos

1. Corte mínimo aleatorizado.
2. Primalidad aleatorizada.

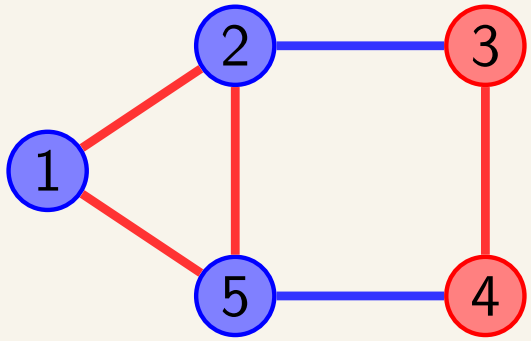
Corte mínimo

Corte

Sea $G = (V, E)$ una gráfica y $\emptyset \subsetneq U \subsetneq V$. El corte $C = \delta(U)$ definido por U es el conjunto de las aristas con un vértice en U y el otro en $V \setminus U$.

Min Cut

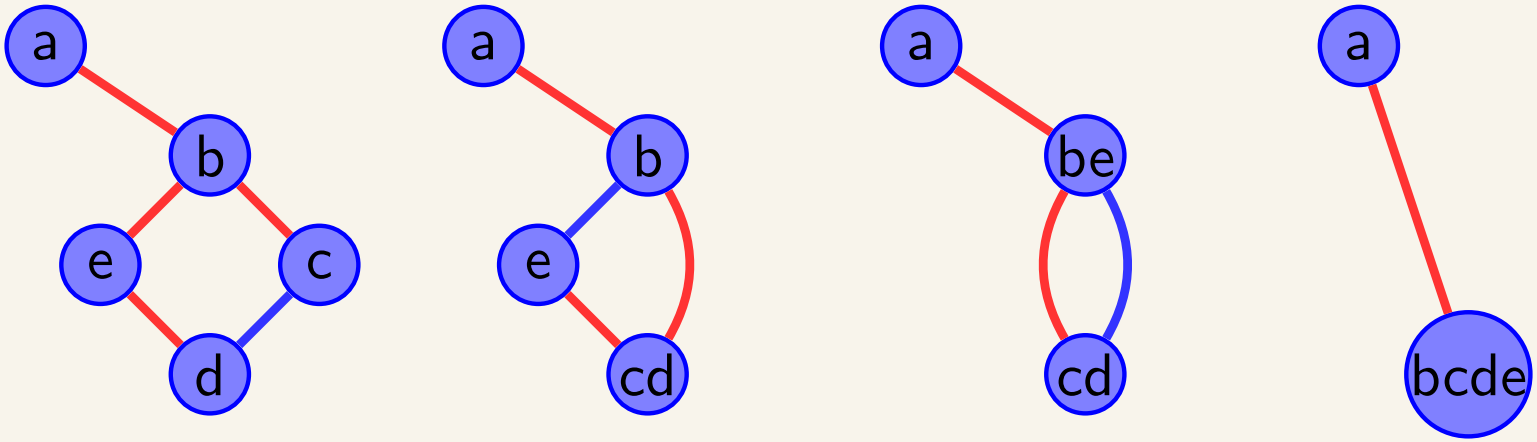
Dada una gráfica G ¿cuál es el tamaño mínimo de sus cortes?



Algoritmo de Karger

Función karger(G)

1. Mientras G tenga al menos 2 vértices[a]:
   1. Escoge aleatoriamente una arista e de G.
   2. Contrae la arista e de G.
2. Escribe G a la salida.



[a] *Global Min-cuts in RNC and Other Ramifications of a Simple Mincut Algorithm*, Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, 1993.

Análisis del algoritmo de Karger

Observaciones

1. Una contracción no disminuye el corte mínimo ni crea cortes.
2. Al final encuentra un corte original de la gráfica.

Probabilidad de no escoger aristas del corte mínimo

Sea C un corte mínimo de G de tamaño k. El grado mínimo de G es al menos k. G tiene al menos $\frac{1}{2}kn$ aristas. Sea E_i el evento de no escoger una arista de C en el paso i. La probabilidad de escoger una arista de C en el primer paso es $\leq \frac{k}{\frac{1}{2}kn} = \frac{2}{n}$ por lo que $P[E_1] \geq 1 - \frac{2}{n}$.

Continuación del análisis

Probabilidad de no escoger aristas del corte mínimo

En general, en el paso i quedan $n - i + 1$ vértices. El tamaño del corte mínimo es al menos k, por lo que la gráfica tiene al menos $\frac{1}{2}k(n - i + 1)$ aristas. Por lo tanto $P[E_i | \cap_{j=1}^{i-1} E_j] \geq 1 - \frac{2}{n-i+1}$. Finalmente

$$P\left[\bigcap_{i=1}^{n-2} E_i\right] \geq \prod_{i=1}^{n-2}\left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)} > \frac{2}{n^2}.$$

Probabilidad de encontrar un corte mínimo

Si ejecutamos el proceso $\frac{n^2}{2}$ veces, la probabilidad de no encontrarlo es

$$< \left(1 - \frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e} \approx 0.367879441.$$

Primalidad

Función fermat(n, k)

1. Repite k veces:
   1. Escoge a aleatoriamente en el intervalo $[1, n-1]$.
   2. Si $a^{n-1} \neq 1 \mod n$ regresa *compuesto*.
2. Regresa *primo*.

Observaciones

1. Cuando el algoritmo dice que n es compuesto nunca se equivoca.
2. Cuando el algoritmo dice que n es primo se puede equivocar.
3. Los números de Carmichael siempre regresan *primo*.
4. El algoritmo de Miller-Rabin no tiene este problema[a].

[a] *Probabilistic algorithm for testing primality*, Journal of Number Theory 12 (1): 128–138, 1980.

Algoritmo de Miller-Rabin

Función rabin(n, k)

Factoriza $n - 1 = 2^s d$ con d impar.

Repite k veces:

1. Escoge a aleatoriamente en el intervalo $[2, n-2]$.
2. $x \leftarrow a^d \mod n$.
3. Si $1 < x < n - 1$ entonces:
   1. Repite s veces:
      1. $x \leftarrow x^2 \mod n$.
      2. Si $x = 1$ regresa *compuesto*.
      3. Si $x = n - 1$ continúa el ciclo externo.
   2. Regresa *compuesto*.

Regresa *primo*.

Teorema

Si dice que n es primo se equivoca con probabilidad menor a 4^{-k}.

Aleatorio y seudoaleatorio

Procesos aleatorios

Los procesos aleatorios son herramientas matemáticas. Es difícil saber si un proceso real es aleatorio (y posiblemente no existan).

Generadores seudoaleatorios

Son procesos deterministas que calculan secuencias que parecen aleatorios pero que eventualmente se repiten. Los más comunes son los generadores de congruencias lineales de la forma $X_{n+1} = (aX_n + b) \mod m$ para algunas constantes a, b y m.

Algoritmos glotones

También conocidos como

1. Algoritmos ávaros.
2. Algoritmos ávidos.
3. Algoritmos golosos.
4. Algoritmos miopes.
5. Algoritmos perezosos.
6. Algoritmos voraces.

¿Por qué?

Toman la decisión que parece la mejor a cada momento.

Fracciones egipcias

Fracciones egipcias

Son representaciones de racionales como sumas de inversos de enteros.

Algoritmo de Fibonacci

Reemplaza la fracción simplificada $\frac{x}{y}$ por la expresión $\frac{1}{\lceil y/x \rceil} + \frac{x-(y \mod x)}{y\lceil y/x \rceil}$.

Ejemplo

El algoritmo de Fibonacci calcularía

$$\frac{5}{121} = \frac{1}{25}+\frac{1}{757}+\frac{1}{763309}+\frac{1}{873960180913}+\frac{1}{1527612795642093418846225}$$

cuando la mejor solución es

$$\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}.$$

Referencias

El libro de Motwani y Raghavan

Randomized Algorithms. Cambridge University Press. 1995.

El libro de Skiena

The Algorithm Design Manual. Springer. 2008.

El libro de Arora y Barak

Computational Complexity: A Modern Approach. Cambridge University Press. 2009.

Contenido

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Complejidad computacional

La teoría de la complejidad computacional es el estudio del costo involucrado en la solución de los problemas. Se desea medir la cantidad de recursos: tiempo, espacio, etc.

Análisis de algoritmos

El análisis de algoritmos es el análisis de los recursos usados por un algoritmo dado. Se desean analizar al menos dos aspectos:

- Cota superior: dar un buen algoritmo.
- Cota inferior: probar que ningún algoritmo es mejor.

Clases de complejidad

La cantidad de recursos se puede ver como una función. Las funciones se pueden catalogar en clases de complejidad:

- Logarítmicas ($\log_2 n$).
- Lineales ($3n + 1$).
- Cuadráticas ($n^2 - n + 1$).
- Polinomiales ($n^4 + n^2 + 1$).
- Exponenciales (2^n).
- Factoriales ($n!$).

De polinomial hacia arriba es bueno. Hacia abajo es malo.

Factores que afectan la eficiencia

Entre otros

- El problema que se esté resolviendo.
- El lenguaje de programación.
- El compilador.
- El hardware.
- La habilidad del programador.
- La efectividad del programador.
- El algoritmo.

Implementación de algoritmos

Proceso de implementación

Considere los siguientes tres procesos:

- Un programador transcribe un algoritmo como un programa.
- Un compilador toma un programa y lo vuelve un ejecutable.
- Una computadora lo ejecuta con una entrada y obtiene una salida.

Factor constante

En ellos aparece un factor constante que depende de varios factores:

- La habilidad y efectividad del programador.
- El lenguaje de programación y el compilador.
- La velocidad y otros recursos del hardware.

Es por eso que nos interesa medir el uso de recursos como una función del tamaño de la entrada ignorando los factores constantes.

Notaciones O, Ω y Θ

Notación O (Omicron)

Diremos que $f(n) \in O(g(n))$ si existen constantes $c \geq 0$ y $n_0 \geq 0$ tales que $f(n) \leq cg(n)$ para toda $n \geq n_0$.

Notación Ω (Omega)

Diremos que $f(n) \in \Omega(g(n))$ si existen constantes $c > 0$ y $n_0 \geq 0$ tales que $f(n) \geq cg(n)$ para toda $n \geq n_0$.

Notación Θ (Zeta)

Diremos que $f(n) \in \Theta(g(n))$ si $f(n) \in O(g(n))$ y $f(n) \in \Omega(g(n))$.

Órdenes comunes

Ejemplos de órdenes y algoritmos

- Constante $O(1)$: determinar si un número es par.
- Logarítmico $O(\log n)$: búsqueda binaria.
- Polilogarítmico $O((\log n)^c)$: decidir si n es primo.
- Lineal $O(n)$: búsqueda lineal, suma de n bits.
- Linearítmico $O(n \log n)$: ordenamiento por mezcla.
- Cuadrático $O(n^2)$: ordenamiento de burbuja.
- Cúbico $O(n^3)$: algoritmo de Warshall.
- Polinomial $O(n^c)$: acoplamiento máximo.
- Exponencial $O(c^n)$: generación de cadenas c-arias.
- Factorial $O(n!)$: generación de permutaciones.

Teoremas de la suma y del producto

Primer teorema de la suma

Si $f_1(n) \in O(g_1(n))$ y $f_2(n) \in O(g_2(n))$ entonces

$$f_1(n) + f_2(n) \in O(g_1(n) + g_2(n)).$$

Segundo teorema de la suma

Si $f_1(n) \in O(g_1(n))$ y $f_2(n) \in O(g_2(n))$ entonces

$$f_1(n) + f_2(n) \in O(\max(g_1(n), g_2(n))).$$

Teorema del producto

Si $f_1(n) \in O(g_1(n))$ y $f_2(n) \in O(g_2(n))$ entonces

$$f_1(n) \cdot f_2(n) \in O(g_1(n) \cdot g_2(n)).$$

Tipos de análisis

Análisis de peor caso

Se desea encontrar el máximo tiempo de ejecución usado por cualquiera de las posibles entradas de cierto tamaño.

Análisis de caso promedio

Se desea encontrar el tiempo de ejecución esperado dada una distribución de probabilidad (generalmente uniforme) de las entradas de cierto tamaño.

Análisis probabilístico

Tiempo de ejecución esperado y la probabilidad de que esto ocurra.

Análisis amortizado

Tiempo de ejecución promedio de una serie de ejecuciones.

Complejidad temporal

Análisis de peor caso

¿Cuánto tiempo toma ejecutar el algoritmo en el peor caso?

- Asignación: $O(1)$.
- Llamar a una función: $O(1)$.
- Salir de una función: $O(1)$.
- Decisión: el tiempo de la prueba más el peor tiempo de las ramas.
- Ciclo: la suma de los tiempos de cada una de las iteraciones.

Tiempo de ejecución de la multiplicación

Función multiplica(y, z)

1. $x \leftarrow 0$.
2. Mientras $z > 0$ haz:
   1. Si z es impar entonces $x \leftarrow x + y$.
   2. $y \leftarrow 2y$.
   3. $z \leftarrow \lfloor z/2 \rfloor$.
3. Regresa x.

Análisis

- Supongamos que y, z tienen n bits.
- La llamada, la asignación y el regreso cuestan $O(1)$ cada una.
- La decisión y las asignaciones dentro del ciclo cuestan $O(1)$ cada una.
- El ciclo se ejecuta un máximo de n veces.
- Por lo tanto todo el proceso toma tiempo $O(n)$.

Tiempo de ejecución del algoritmo de burbuja

Función burbuja(A, n)

1. Para $i \leftarrow 1$ a $n-1$ haz:
   1. Para $j \leftarrow 1$ a $n-i$ haz:
      1. Si $A_j > A_{j+1}$ entonces intercambia A_j con A_{j+1}.

Análisis

- La llamada, la decisión y el regreso cuestan $O(1)$ cada una.
- El ciclo interno se ejecuta $n-i$ veces y cuesta $O(n-i)$.
- El ciclo externo cuesta

$$O\left(\sum_{i=1}^{n-1}(n-i)\right) = O\left(\sum_{i=1}^{n-1} i\right) = O\left(\frac{n(n-1)}{2}\right) = O(n^2).$$

Ordenamiento por mezcla

Función ordenamezcla(L, n)

1. Si $n \leq 1$ entonces regresa L.
2. Parte L en dos listas L_1 y L_2 del mismo tamaño ($n_1 = n_2 = n/2$).
3. Regresa mezcla(ordenamezcla(L_1, n_1), ordenamezcla(L_2, n_2)).

Suposiciones

- Que n es una potencia de 2.
- Que la función mezcla puede mezclar dos listas ordenadas con un total de n elementos en tiempo $O(n)$.
- Sea $T(n)$ el tiempo de ejecución de ordenamezcla(L, n). Entonces

$$T(n) = \begin{cases} c & \text{si } n = 1 \\ 2T(n/2) + dn & \text{en otro caso} \end{cases}$$

para algunas constantes c y d.

Tiempo de ejecución

Sustitución repetida

- Suponga que $n > 1$. Entonces:

$$\begin{aligned} T(n) &= 2T(n/2) + dn \\ &= 2(2T(n/4) + dn/2) + dn \\ &= 4T(n/4) + 2dn \\ &= 4(2T(n/8) + dn/4) + 2dn \\ &= 8T(n/8) + 3dn. \end{aligned}$$

- Aparece el patrón $T(n) = 2^i T(n/2^i) + idn$.
- Tomando $i = \log_2 n$ se tiene que

$$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + dn \log_2 n = dn \log_2 n + cn.$$

- Por lo tanto $T(n) \in O(n \log n)$.

Un teorema general

Teorema

Si n es una potencia de c entonces la solución a la recurrencia

$$T(n) = \begin{cases} d & \text{si } n = 1 \\ aT(n/c) + bn & \text{si } n > 1 \end{cases}$$

está dada por:

- $T(n) \in O(n)$ si $a < c$,
- $T(n) \in O(n \log n)$ si $a = c$ o
- $T(n) \in O(n^{\log_c a})$ si $a > c$.

Detalles a considerar

Modelos de cómputo

¿Qué significa que el algoritmo de la multiplicación sea $O(n)$?

- Supusimos que cada suma de n bits toma tiempo $O(1)$.
- Esto sólo es cierto en el modelo de palabras.
- En el modelo de bits la misma suma toma tiempo $O(n)$.

La verdad, toda la verdad y...

El algoritmo de la multiplicación también toma tiempo $O(n^2)$.

Factor constante

Hay que tener una idea de cuánto vale el factor constante.

- No es lo mismo hacer $2n$ operaciones que $2^{2^{2^{2^2}}}$ n operaciones.
- Un factor muy grande puede volver impráctico un algoritmo.

Algoritmos y problemas

Observaciones sobre las notaciones

Las notaciones O, Ω y Θ significan cosas distintas cuando se le aplican a algoritmos o a problemas.

Ejemplo con el algoritmo de burbuja

El algoritmo de burbuja toma tiempo $O(n^2)$, pero ¿es lo mejor que puedo decir? Tal vez soy demasiado flojo como para descubrirlo o tal vez no se sabe. Lo mejor que se puede decir es que toma tiempo $\Theta(n^2)$.

Ejemplo con el problema de ordenamiento

El problema de ordenamiento se puede resolver en tiempo $O(n \log n)$, pero ¿existe un algoritmo más rápido?

Cota inferior

Ordenamiento con comparaciones

- Todo algoritmo de ordenamiento debe ordenar su entrada.
- La entrada puede venir ordenada de $n!$ formas distintas.
- El algoritmo no sabe cuál de esas entradas es.
- Como cada comparación sólo puede tener dos valores posibles, el espacio de búsqueda se reduce a lo mucho a la mitad con cada una.
- Por lo tanto, si el algoritmo puede ordenar en un máximo de $T(n)$ comparaciones entonces

$$2^{T(n)} \geq n!$$

- Es decir $T(n) \geq \log n! \in \Theta(n \log n)$

Complejidad y tamaño de la entrada

Complejidad

Anteriormente dijimos que la complejidad es una función que mide el uso de los recursos de un algoritmo.

Tamaño de la entrada

Lo que no hemos dicho es que la función es en términos del tamaño de la entrada, lo que nos lleva a una pregunta.

¿Cómo expresamos la entrada y la salida?

Codificaciones

Dependiendo de cómo expresemos un algoritmo, debemos codificar la entrada y la salida de alguna manera.

Ejemplo

Un número natural se puede codificar de varias formas:

1. En español: doce.
2. En decimal: 12.
3. En binario: 1100.
4. En hexadecimal: D.
5. En unario: 111111111111.

Análisis del algoritmo de Euclides

¿Cuántas divisiones hace?

El peor caso es cuando A y B son dos números de Fibonacci consecutivos. En particular hace n divisiones si $A = F_{n+2}$ y $B = F_{n+1}$.

¿Cuál es el tamaño de la entrada?

El número de dígitos de B depende de la codificación:

1. En decimal $\log_{10} B \approx \log_{10} \phi^n \approx \frac{n}{5}$.
2. En binario $\log_2 B \approx \log_2 \phi^n \approx \frac{2n}{3}$.
3. En unario $B \approx \phi^n \approx 1.6^n$.

Buenas codificaciones

¿Qué codificación usar?

En general, aquella que requiera la menor cantidad de símbolos (bits) para representar a los objetos correspondientes.

Ejemplos

1. El natural n se representa en binario usando $\lceil \log_2(n+1) \rceil$ bits.
2. El entero n se representa en binario usando $1 + \lceil \log_2(|n|+1) \rceil$ bits.
3. El vector $(x_1, x_2, \ldots, x_n)$ de enteros se representa usando $n + \lceil \log_2(n+1) \rceil + \sum_{i=1}^{n} \lceil \log_2(|x_i|+1) \rceil$ bits.

Tiempo polinomial

Algoritmos polinomiales

Un algoritmo se dice polinomial si su tiempo de ejecución es $O(p(n))$ para algún polinomio p en el tamaño n de su entrada.

La clase P

Un problema de decisión se dice polinomial si existe algún algoritmo polinomial que lo resuelva. El conjunto de estos problemas es la clase P.

Tiempo polinomial no determinista

Algoritmos polinomiales no deterministas

Un algoritmo no determinista se dice polinomial si su tiempo de ejecución es $O(p(n))$ para algún polinomio p en el tamaño n de su entrada.

La clase NP

Un problema de decisión se dice polinomial no determinista si existe algún algoritmo polinomial no determinista que lo resuelva. El conjunto de estos problemas es la clase NP.

Problemas NP completos

Problemas NP completos

Un problema de decisión $\mathcal{P}$ se dice NP completo si:

1. Está en NP.
2. Todo problema en NP se puede reducir a $\mathcal{P}$ en tiempo polinomial.

Problemas NP duros

Un problema $\mathcal{P}$ se dice NP duro si todo problema en NP se puede reducir a $\mathcal{P}$ en tiempo polinomial.

La pregunta del millón

¿P = NP?

El Instituto Clay de Matemáticas[a] pagará 1 000 000 US a quien responda correctamente a esta pregunta.

[a] Las reglas están en `http://www.claymath.org/millennium/`

Problemas NP completos

Problemas descritos anteriormente

1. Problema de satisfacibilidad.
2. Problema de 3-satisfacibilidad.
3. Ciclo hamiltoniano.
4. Max Clique: ¿existirá un clan de G de tamaño $\geq k$?
5. Min Color: ¿existirá una coloración de G con $\leq k$ colores?
6. Max Cut: ¿existirá un corte de G de tamaño $\geq k$?
7. Muchos más.

Errores comunes

Todas las siguientes son probablemente falsas

1. Los problemas NP completos son los más difíciles que se conocen.
2. Los problemas NP completos son difíciles porque tienen muchas soluciones.
3. Resolver problemas NP completos requiere tiempo exponencial.
4. Todas las instancias de los problemas NP completos son difíciles.

Otras clases de complejidad

Clases de problemas de decisión

- ZPP Se pueden resolver sin error en tiempo polinomial con algoritmos aleatorizados.
- RP Se pueden resolver con error unilateral en tiempo polinomial con algoritmos aleatorizados.
- BPP Se pueden resolver con error bilateral en tiempo polinomial con algoritmos aleatorizados.

Clases de problemas de conteo

- $\sharp$P Asociados a problemas de decisión en NP.
- $\sharp$PC Los $\sharp$P se pueden reducir en tiempo polinomial a estos.

Otras clases de complejidad

Clases de algoritmos

1. Un algoritmo es fuertemente polinomial si todas sus entradas son enteros, se tarda tiempo polinomial en la cantidad de enteros en la entrada (cada operación aritmética toma tiempo unitario) y usa espacio polinomial en el tamaño de la entrada.
2. Un algoritmo es débilmente polinomial si es polinomial pero no es fuertemente polinomial.
3. Un algoritmo es casi polinomial si no es polinomial pero se ejecuta en tiempo $2^{O((\log n)^c)}$ para alguna constante c.
4. Un algoritmo es seudopolinomial si se ejecuta en tiempo polinomial en el valor de sus entradas.

¿Cómo lidiar con los problemas NP completos?

Estrategias posibles

1. Heurísticas.
2. Restricción.
3. Parametrización.
4. Aleatorización.
5. Aproximación.

Referencias

El libro de Arora y Barak

Computational Complexity: A Modern Approach, Cambridge University Press, 2009.

El libro de Garey y Johnson

Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.

El libro de Parberry

Problems in Algorithms, Prentice Hall, 1995.

Contenido

Problemas de optimización NP

Definición

Un problema de optimización NP $\mathcal{P} : \mathcal{I} \to \mathcal{S}$ cumple que:

1. Su conjunto $\mathcal{I}$ de instancias se puede reconocer en tiempo polinomial (es decir, el problema de decisión $e \in \mathcal{I}$ está en P).
2. Si $e \in \mathcal{I}$ y $\mathcal{S}(e)$ es su conjunto de soluciones entonces:
   1. Existe un polinomio p tal que si $s \in \mathcal{S}(e)$ entonces $|s| \leq p(|e|)$.
   2. Si $|t| \leq p(|e|)$ se puede decidir en tiempo polinomial si $t \in \mathcal{S}(e)$.
3. La función objetivo $m : \mathcal{I} \times \mathcal{S} \to \mathcal{Z}$ se puede calcular en tiempo polinomial.

Las clases NPO y NP completo

Definición

La clase NPO es el conjunto de todos los problemas de optimización NP.

NPO y NP completo

Si un problema $\mathcal{P} \in$ NPO se puede resolver en tiempo polinomial, entonces su versión de decisión $\mathcal{D}$ también se puede resolver en tiempo polinomial.

Conclusión

Si P $\neq$ NP y $\mathcal{D}$ es NP completo, entonces $\mathcal{P}$ no se puede resolver en tiempo polinomial. En este caso sacrificamos la optimalidad y buscamos soluciones aproximadas que se puedan calcular en tiempo polinomial.

Desempeño de una solución

Definición

Sea $e \in \mathcal{I}$ y $s \in \mathcal{S}(e)$. Entonces, el desempeño de s con respecto a e está dado por

$$R(e, s) = \frac{m(e, s)}{\mathcal{P}(e)},$$

es decir, la razón entre el valor de s y el valor de una solución óptima.

Observación

Note que $R(e, s) \leq 1$ para un problema de maximización y $R(e, s) \geq 1$ para un problema de minimización.

Desempeño de un algoritmo

Definición

Sea $\mathcal{P} \in$ NPO de minimización, sea $\mathcal{A}$ un algoritmo que para toda $e \in \mathcal{I}$ regresa $\mathcal{A}(e) \in \mathcal{S}(e)$ y sea $\alpha : \mathcal{N} \to \mathcal{R}^+$. Si $\mathcal{A}$ cumple que

$$R(e, \mathcal{A}(e)) \leq \alpha(|e|)$$

para toda $e \in \mathcal{I}$ entonces diremos que es un algoritmo de aproximación para $\mathcal{P}$ con garantía α. Si además $\mathcal{A}$ se ejecuta en tiempo polinomial, diremos que $\mathcal{P}$ se puede aproximar con garantía α.

Observación

Para $\mathcal{P} \in$ NPO de maximización se cambia la desigualdad por

$$R(e, \mathcal{A}(e)) \geq \alpha(|e|).$$

Casa abierta al tiempo Azcapotzalco

Aproximación constante

La clase APX

Un problema $\mathcal{P} \in$ NPO pertenece a la clase APX si se puede aproximar con alguna garantía α constante.

Observación

Note que $\alpha \leq 1$ para un problema de maximización y $\alpha \geq 1$ para un problema de minimización.

Esquemas de aproximación

Sea $\mathcal{P} \in$ NPO de minimización y sea $\mathcal{A}$ un algoritmo que para toda $e \in \mathcal{I}$ y todo racional $\alpha > 1$ regresa $\mathcal{A}(e, \alpha) \in \mathcal{S}(e)$ tal que $R(e, \mathcal{A}(e, \alpha)) \leq \alpha$. Entonces $\mathcal{A}$ es un esquema de aproximación para $\mathcal{P}$.

La clase PTAS

Un problema $\mathcal{P} \in$ NPO pertenece a la clase PTAS si tiene un esquema de aproximación polinomial en $|e|$.

La clase FPTAS

Un problema $\mathcal{P} \in$ NPO pertenece a la clase FPTAS si tiene un esquema de aproximación polinomial en $|e|$ y $\frac{1}{1-\alpha}$.

Observaciones

1. Definiciones análogas para maximización.
2. En un PTAS no importa la dependencia de α.
3. FPTAS $\subseteq$ PTAS $\subseteq$ APX $\subseteq$ NPO (estrictas si y sólo si P $\neq$ NP).

Reducibilidad

Sean A y B dos problemas en NPO. Diremos que A es PTAS reducible a B si existen tres funciones f, g, c tales que:

1. Para toda $x \in \mathcal{I}_A$ y todo racional $\alpha > 1$ se puede calcular $f(x, \alpha) \in \mathcal{I}_B$ en tiempo polinomial respecto a $|x|$.
2. Para toda $x \in \mathcal{I}_A$, todo racional $\alpha > 1$ y toda $y \in \mathcal{S}_B(f(x, \alpha))$ se puede calcular $g(x, y, \alpha) \in \mathcal{S}_A(x)$ en tiempo polinomial en $|x|$ y $|y|$.
3. La función c se puede calcular e invertir (en tiempo polinomial).
4. Para toda $x \in \mathcal{I}_A$, todo racional $\alpha > 1$ y toda $y \in \mathcal{S}_B(f(x, \alpha))$:

$$R_B(f(x, \alpha), y) \leq c(\alpha) \text{ implica } R_A(x, g(x, y, \alpha)) \leq \alpha.$$

Dos teoremas de cerradura

Teorema de cerradura APX

Si A es PTAS reducible a B y B está en APX, entonces A está en APX.

Teorema de cerradura PTAS

Si A es PTAS reducible a B y B está en PTAS, entonces A está en PTAS.

Problemas NPO y APX completos

Problemas NPO completos

Un problema $\mathcal{P} \in$ NPO es NPO completo si todo problema $\mathcal{Q} \in$ NPO es PTAS reducible a $\mathcal{P}$.

Problemas APX duros

Un problema $\mathcal{P} \in$ NPO es APX duro si todo problema $\mathcal{Q} \in$ APX es PTAS reducible a $\mathcal{P}$.

Problemas APX completos

Un problema $\mathcal{P} \in$ APX es APX completo si todo problema $\mathcal{Q} \in$ APX es PTAS reducible a $\mathcal{P}$.

Satisfacibilidad

Max Sat

Es aproximable con garantía 0.770[a] pero es APX completo[b].

---

[a] Asano, Hori, Ono, Hirata, *A theoretical framework of hybrid approaches to MAX SAT*, LNCS 1350/1997, 153-162, 1997.

[b] Papadimitriou, Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci. 43, 425-440, 1991.

Max 3Sat

Es aproximable con garantía 0.801[a] pero es APX completo.

---

[a] Trevisan, Sorkin, Sudan, Williamson, *Gadgets, approximation, and linear programming*, Proc. 37th Ann. IEEE Symp. on Foundations of Comput. Sci., IEEE Computer Society, 617-626, 1996.

Hamiltonicidad

Max Camino

Dada una gráfica $G = (V, E)$ encontrar la longitud máxima de un camino simple es aproximable con garantía $\frac{\log |V|}{|V|}$ pero no está en APX[a].

[a] Alon, Yuster, Zwick, *Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs*, Proc. 26th Ann. ACM Symp. on Theory of Comp., ACM, 326-335, 1994.

Agente viajero

Dada una gráfica completa $G = (V, E)$ con costos en las aristas encontrar la longitud mínima de un ciclo hamiltoniano es NPO completo, aunque es aproximable con garantía $\frac{3}{2}$ si los costos son métricos[a].

[a] Christofides, *Worst-case analysis of a new heuristic for the travelling salesman problem*, TR 388, Carnegie-Mellon University, 1976.

Clan máximo y coloración mínima

Max Clan

Es aproximable[a] con garantía $O(\frac{(\log |V|)^2}{|V|})$ pero no es aproximable con garantía $|V|^{-1+\epsilon}$ para toda $\epsilon > 0$ a menos que NP=ZPP.

[a]Boppana, Halldórsson, *Approximating maximum independent sets by excluding subgraphs*, Bit 32, 180-196, 1992.

Min Color

Es aproximable[a] con garantía $O(\frac{|V|(\log \log |V|)^2}{(\log |V|)^3})$ pero no es aproximable con garantía $|V|^{1-\epsilon}$ para toda $\epsilon > 0$ a menos que NP=ZPP.

[a]Halldórsson, *A still better performance guarantee for approximate graph coloring*, Inform. Process. Lett. 45, 19-23, 1993.

Corte máximo

Max Cut

Es aproximable con garantía 0.878[a] pero es APX completo[b] y no es aproximable con garantía 0.941[c].

---

[a] Goemans, Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM 42, 1115-1145, 1995.

[b] Papadimitriou, Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci. 43, 425-440, 1991.

[c] Håstad, *Some optimal inapproximability results*, Proc. 29th Ann. ACM Symp. on Theory of Comp., ACM, 1-10, 1997.

Cubierta y empaquetamiento de conjuntos

Max Set Packing

Los mismos resultados que Max Clan.

Min Set Cover

Es aproximable[a] con garantía $1 + \ln |S|$ pero no es aproximable[b] con garantía $c \log |S|$ para alguna $c > 0$.

---

[a] Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci. 9, 256-278, 1974.

[b] Raz, Safra, *A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP*, Proc. 29th Ann. ACM Symp. on Theory of Comp., ACM, 475-484, 1997.

Agente viajero con costos generales

Agente viajero

Dada una gráfica completa $G = (V, E)$ con costos no negativos en las aristas, encontrar la longitud mínima de un ciclo hamiltoniano no se puede aproximar en tiempo polinomial (a menos que P=NP).

Idea de la demostración

Si se pudiera aproximar con cualquier garantía $\alpha = \alpha(|V|) \geq 1$ entonces se podría resolver el problema del ciclo hamiltoniano en tiempo polinomial.

Reducción

Sea $H = (V, F)$ una gráfica cualquiera. Sea $G = (V, E)$ una gráfica completa con costos en las aristas $c_e = 1$ si $e \in F$ y $c_e = \alpha|V|$ si $e \notin F$.

Agente viajero con costos generales

Observaciones sobre ciclos hamiltonianos

1. Un ciclo hamiltoniano de G tiene costo $|V|$ si sólo usa aristas de H y tiene costo $\geq \alpha|V| + |V| - 1$ en caso contrario.
2. Si H es hamiltoniana, el costo mínimo es $|V|$ y si no lo es el costo mínimo es $\geq \alpha|V| + |V| - 1$.

Contradicción

Si el supuesto algoritmo de aproximación existiera, entonces generaría un ciclo hamiltoniano de G de costo $\leq \alpha|V|$ cuando H es hamiltoniana y de costo $\geq \alpha|V| + |V| - 1$ en caso contrario.

Agente viajero con costos métricos

Agente viajero métrico

Dada una gráfica completa $G = (V, E)$ con costos *métricos* en las aristas, encontrar la longitud mínima de un ciclo hamiltoniano se puede aproximar en tiempo polinomial con garantía $\frac{3}{2}$.

Costos métricos

Si $u, v, w \in V$ entonces $c_{uv} + c_{vw} \geq c_{uw}$.

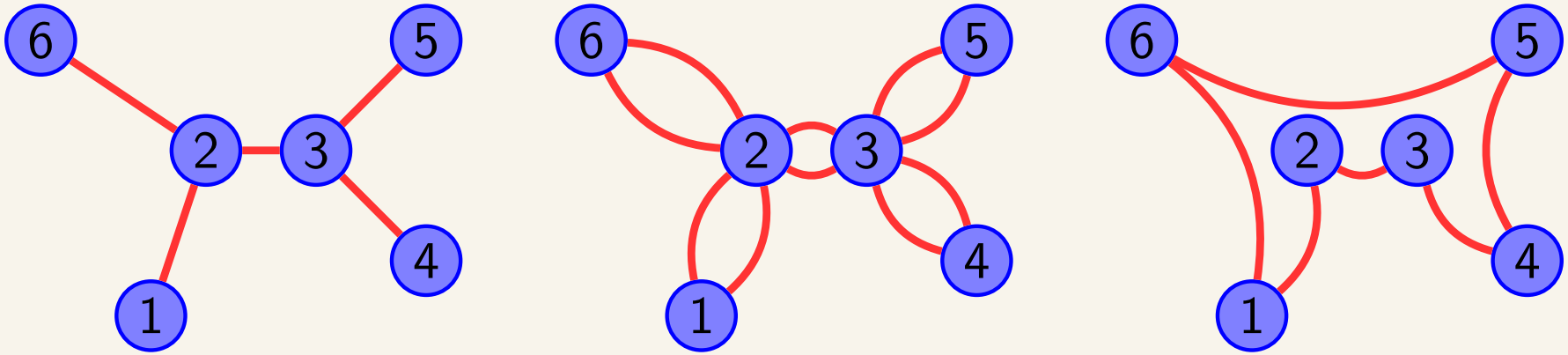
Simplificación

Vamos a presentar un algoritmo de aproximación con garantía 2, que resulta de una simplificación del algoritmo de Christofides.

Algoritmo simplificado

Entrada G y c

1. Construye un árbol abarcador T de costo mínimo en (G, c).
2. Duplica todas las aristas de T para obtener una gráfica $2T$.
3. Encuentra un circuito euleriano C de $2T$.
4. Convierte C en un circuito hamiltoniano H de G *saltando* vértices.



Garantía de aproximación

Demostración

Sea H^* un circuito hamiltoniano de costo mínimo de (G, c). Como H^* contiene un *árbol abarcador* se tiene que

$$c(T) \leq c(H^*).$$

Por la condición métrica $c(H) \leq c(2T)$ y entonces se cumple que

$$c(H^*) \leq c(H) \leq c(2T) = 2c(T) \leq 2c(H^*).$$

Christofides

Se completa T a una gráfica euleriana de forma óptima.

Cubierta por conjuntos

Min Set Cover

Sea S un conjunto finito y $\mathcal{F}$ una familia de subconjuntos de S. Una subfamilia $\mathcal{E} \subseteq \mathcal{F}$ es una cubierta de tamaño $|\mathcal{E}|$ si la unión de los elementos de $\mathcal{E}$ es S. ¿Cuál es el tamaño mínimo de una cubierta de $\mathcal{F}$?

Garantía

Un algoritmo glotón produce una solución factible con garantía $H(|S|)$ donde

$$H(n) = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$$

incluso si cada elemento $f \in \mathcal{F}$ tiene un costo asociado $c(f)$.

Algoritmo glotón

Algoritmo de Chvátal

1. Sea $T \leftarrow \emptyset$ el subconjunto de los elementos cubiertos.
2. Mientras $T \neq S$:
   1. Escoja un conjunto $f \in \mathcal{F}$ que minimice $\alpha(f, T) = \frac{c(f)}{|f \setminus T|}$.
   2. Para cada $i \in f \setminus T$ sea $\beta_i \leftarrow \alpha(f, T)$.
   3. Sea $T \leftarrow T \cup f$.
3. Emite a la salida todos los conjuntos escogidos.

Análisis

Usa dualidad de programación lineal[a].

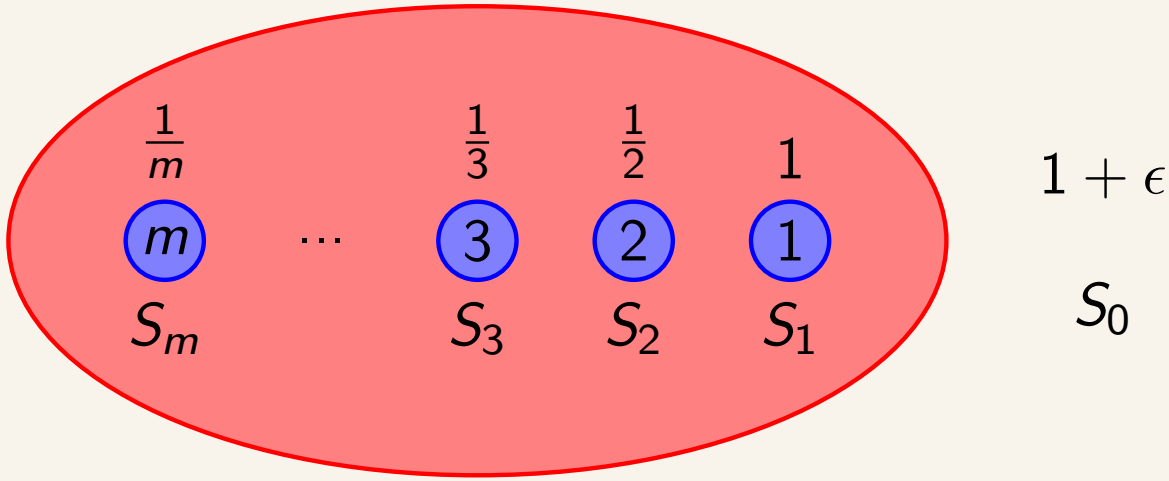
---

[a] Chvátal, *A Greedy Heuristic for the Set-Covering Problem*. Mathematics of Operations Research 4:3, 233-235, 1979.

Optimalidad del resultado

Ejemplo justo

Para toda m considere la familia de $m+1$ subconjuntos $S_0 = \{1, \ldots, m\}$ con $c_0 = 1 + \epsilon$ y $S_i = \{i\}$ con $c_i = \frac{1}{i}$ para toda $1 \leq i \leq m$.



El algoritmo glotón escoge estos conjuntos en el orden $S_m, \ldots, S_1$ con costo $H(m)$ mientras que lo óptimo es escoger S_0.

Corte máximo

Weighted Max Cut

Dada una gráfica G con costos en sus aristas ¿cuál es el costo máximo de sus cortes?

Algoritmos de aproximación

Hay al menos cuatro con garantía 0.5:

1. Búsqueda local.
2. Aleatorizado.
3. Esperanza condicional.
4. Espacio de búsqueda pequeño.

Y otro (basado en programación semidefinida) con garantía 0.878.

Algoritmo de búsqueda local

Algoritmo

Mientras no se llegue a un óptimo local, mueve un vértice entre S y $V \setminus S$ si es que produce una mejora.

Prueba

Si $S \subseteq V$ define un óptimo local entonces para toda $v \in S$

$$c(\delta(v) \cap \delta(S)) \geq c(\delta(v) \cap \gamma(S)).$$

Como

$$c(\delta(S)) = \frac{1}{2} \sum_{v \in V} c(\delta(v) \cap \delta(S))$$

tenemos que

$$c(\delta(S)) \geq \frac{1}{2} \sum_{v \in V} \frac{1}{2} c(\delta(v)) = \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} c(S^*).$$

Algoritmo aleatorizado

Algoritmo

Escoge $S \subseteq V$ uniformemente. Esto es equivalente a decidir de manera independiente si $v \in S$ con probabilidad $\frac{1}{2}$.

Prueba

$$\begin{aligned} E[c(\delta(S))] &= \sum_{ij \in E} w_{ij} P[ij \in \delta(S)] \\ &= \frac{1}{2} \sum_{e \in E} w_e \\ &\geq \frac{1}{2} c(S^*). \end{aligned}$$

Algoritmo desaleatorizado

Algoritmo

Numere los vértices $v_1, \ldots, v_n$ y observe que

$$\begin{aligned} E[c(\delta(S))] &= E[c(\delta(S))|v_1 \in S]P[v_1 \in S] + E[c(\delta(S))|v_1 \notin S]P[v_1 \notin S] \\ &\leq \max(E[c(\delta(S))|v_1 \in S], E[c(\delta(S))|v_1 \notin S]). \end{aligned}$$

Suponga que el máximo se alcanza cuando $v_1 \in S$, entonces

$$\begin{aligned} E[c(\delta(S))] &\leq E[c(\delta(S))|v_1 \in S, v_2 \in S]P[v_2 \in S] \\ &\quad + E[c(\delta(S))|v_1 \in S, v_2 \notin S]P[v_2 \notin S] \\ &\leq \max(E[c(\delta(S))|v_1 \in S, v_2 \in S], E[c(\delta(S))|v_1 \in S, v_2 \notin S]). \end{aligned}$$

Si se continúa de esta manera, se obtiene una sucesión de decisiones para $v_1, \ldots, v_n$ que escogen S de modo que $c(\delta(S)) \geq \frac{1}{2}c(S^*)$.

Algoritmo con espacio de búsqueda pequeño

Idea

Para cada $v \in V$ defina la variable aleatoria $X_v = 1$ si $v \in S$ y $X_v = -1$ si $v \notin S$. Entonces $P[ij \in \delta(S)] = P[X_i X_j = -1]$ y para que esto sea igual a $\frac{1}{2}$ sólo se necesita que X_i y X_j sean independientes. Así que basta un método que asigne probabilidades $P[S]$ a cada $S \subseteq V$ de modo que:

1. $P[v \in S] = \frac{1}{2}$ para toda $v \in V$.
2. $P[u \in S, v \notin S] = \frac{1}{4}$ para toda $u \neq v$.
3. Pocas $P[S]$ son positivas.

Esto se puede lograr usando las matrices de Hadamard definidas por

$$H_1 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \text{ y } H_{k+1} = \begin{pmatrix} H_k & H_k \\ H_k & -H_k \end{pmatrix}$$

para cada $k \geq 1$. Esto genera $n + 1 = 2^k$ cortes que revisar.

El problema de la mochila

Definición

Dada una mochila de capacidad M y n objetos con tamaños $t_1, \ldots, t_n$ y valores $v_1, \ldots, v_n$ se desea encontrar un subconjunto de objetos con tamaño total $\leq M$ y máximo valor total.

Solución

Se puede resolver exactamente en tiempo seudopolinomial $O(n^2 v)$ donde $v = \max_{1 \leq i \leq n} v_i$ usando programación dinámica. De hecho, la versión de decisión de este problema es NP completa.

FPTAS para el problema de la mochila

Escalamiento

Dada $\epsilon > 0$, sea $k = \frac{\epsilon v}{n}$. Para toda $1 \leq i \leq n$ defina $w_i = \lfloor \frac{v_i}{k} \rfloor$. Observe que $w = \lfloor \frac{v}{k} \rfloor = \lfloor \frac{n}{\epsilon} \rfloor$. Ejecute el algoritmo de programación dinámica con los valores $w_1, \ldots, w_n$ y emita a la salida el conjunto S^* obtenido o el objeto de mayor valor.

Teorema

El algoritmo anterior se ejecuta en tiempo $O(n^3 \epsilon^{-1})$ y tiene valor al menos $(1 - \epsilon)$ del óptimo del problema original.

Referencias

El libro de Hochbaum

Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, 1997.

El libro de Vazirani

Approximation Algorithms, Springer, 2001.

El libro de Williamson y Shmoys

The Design of Approximation Algorithms, Cambridge University Press, 2011.

Colofón

Voltaire, *La Bégueule*, 1772

Dans ses écrits, un sàge Italien dit que le mieux est l'ennemi du bien.

Lewis Carroll, *Through the Looking-Glass*, 1871

"In our country," said Alice, still panting a little, "you'd generally get to somewhere else if you run very fast for a long time, as we've been doing."

"A slow sort of country!" said the Queen. "Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!"