Tema 5: Complejidad de Problemas de Optimización Aproximados

Serafín Moral

Modelos Avanzados de Computación - Universidad de Granada

Contenido

- Problemas de Optimización
- Algoritmos δ-aproximados
- Análisis de problemas: cubrimiento por vértices, viajante de comercio, corte máximo, mochila, satisfacción máxima.
- Esquema de aproximación polinómico.
- Clase NPO y PO
- Clases APX, PTAS y FPTAS
- Algoritmos pseudo-polinómicos
- Problemas completos

Bibliografía

- G. Ausiello, P. Creszendi et al. (1999) *Complexity and Approximation* . Springer-Verlag, Berlin.
- A Compendium of NP Optimization Problems. http://www.nada.kth.se/theory/compendium

Problemas de Optimización

Problema de Optimización (minimización)

Tenemos unos datos x.
Estos datos tienen asociados un conjunto de soluciones factibles $F(x)$.
$\forall s \in F(x)$ tenemos una función $C(s)$ que evalúa el coste de esta solución.
El problema es encontar un elemento s^* tal que

$$C(s^*) = \min_{s \in F(x)} C(s)$$

Este problema se conoce cómo la versión *constructiva* del problema de optimización.
Si sólo se trata de calcular $C(s^*)$, tendremos la versión de *evaluación*.
En general, son de dificultad similar.

El Problema del Cubrimiento por Vértices

Ejemplo

- Datos: un grafo G no dirigido.
- Soluciones factibles $F(G)$: conjunto de los cubrimientos por vértices de G el conjunto de todos los subconjuntos de vértices A tales que toda arista tiene, al menos un extremo en A.
- Coste de una solución factible A: su número de vértices.

Queremos encontrar el cubrimiento por vértices con un número menor de vértices.

Problemas de Optimización: Maximización

Algunas veces, en vez de tener una función de coste, tenemos una función de albluebeneficio B y tratamos de maximizarla.
Cada ejemplo x, tiene asociado un conjunto de soluciones factibles $F(x)$.
$\forall s \in F(x)$ tenemos una función $B(s)$ que evalúa el beneficio de esta solución.
El problema es encontar un elemento s^* tal que

$$B(s^*) = \max_{s \in F(x)} B(s)$$

Características

- Estos problemas suelen ser equivalentes bajo reducción Turing a los problemas de decisión asociados (ver la reducción del problema del viajante de comercio en el tema del cálculo de funciones) cuando el problema de decisión es NP-completo.
- En este tema no nos vamos a preocupar de resolverlos de forma exacta.
- Vamos a ver problemas que son de similar dificultad cuando se resuelven de forma exacta, son bastante diferentes cuando intentamos aproximarlos: unos no se pueden aproximar con ningún error; otros se pueden aproximar con algún error, pero no con errores muy pequeños; y otros se pueden aproximar con errores arbitrariamente pequeños.

Razón de Eficacia (maximización)

Supongamos un problema de optimización que para una entrada x tiene un óptimo $OPT(x)$ y ALG un algoritmo aproximado que para una entrada x obtiene una solución factible $ALG(x)$ con un coste (beneficio) de $C(ALG(x))$ ($B(ALG(x))$).

Razón de eficacia (maximización)

Un problema de maximización tiene una razón δ si y solo si

$$\delta \geq \frac{OPT(x)}{B(ALG(x))}$$

Buscamos algoritmos polinómicos que tengan una razón de eficacia tan cercana a 1 como sea posible.

Razón de Eficacia (minimización)

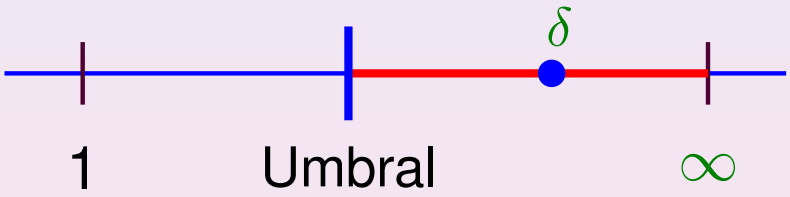
Razón de eficacia (minimización)

Un problema de minimización tiene una razón δ si y solo si

$$\delta \geq \frac{C(ALG(x))}{OPT(x)}$$

Umbral de Aproximación

Umbral de Aproximación: Ínfimo de la razón de eficacia mediante algoritmos polinómicos.



Valor de la razón de eficacia

- Una razón de 1 es equivalente a un algoritmo exacto.
- Puede haber problemas en los que el umbral de aproximación sea infinito (no hay algoritmos polinómicos con una razón de eficacia δ).

Aprox. para Cubrimiento por Vértices

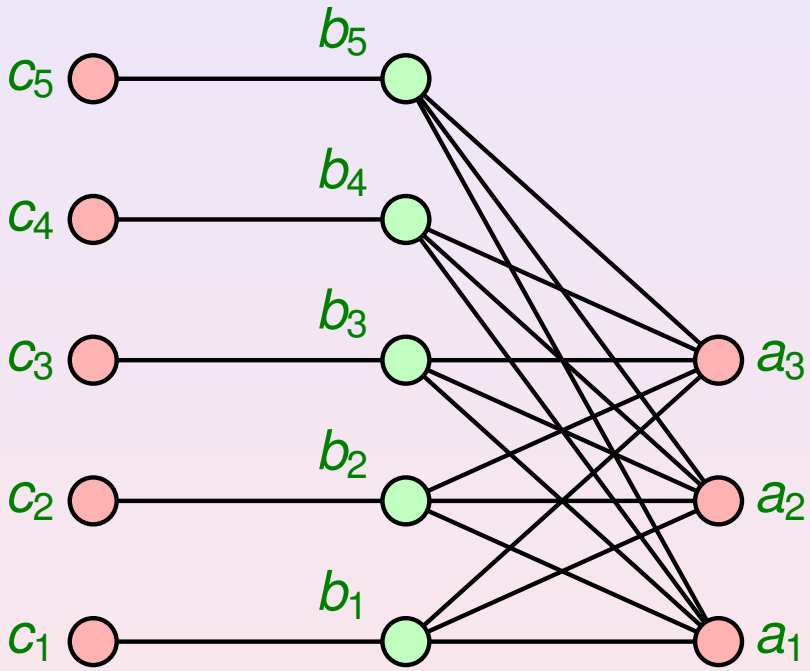
Buscamos el conjunto C con un número mínimo de nodos que sea un cubrimiento para $G = (V, E)$.

Algoritmo Aproximado:

- $C = \emptyset$
- Mientras haya aristas en G
  - Elegir un nodo de G con grado maximo
  - Añadir el nodo a C
  - Borrar de G ese nodo y todos sus enlaces

Este no es un algoritmo δ-aproximado ningún valor de δ.
El error puede llegar a ser de orden $\log(n)$.

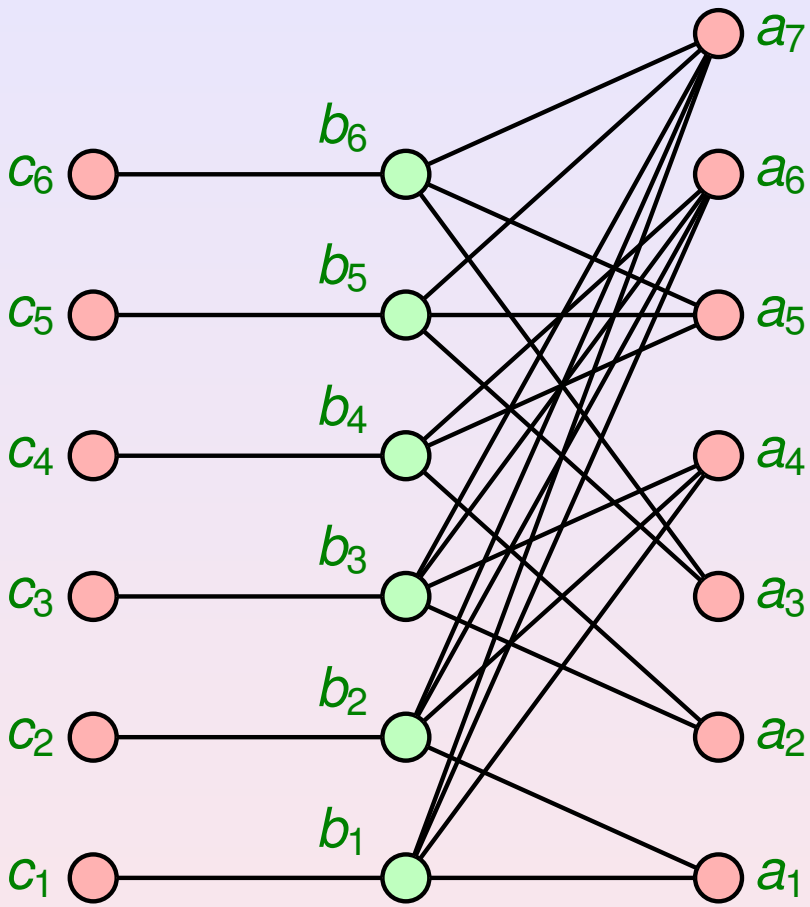
Cubrimiento por Vértices



La heurística podría elegir los nodos rojos.

Con los verdes sería suficiente

Se puede equivocar más



En los nodos de la derecha se sigue el siguiente procedimiento: Para i desde 2 a 5 (uno menos que los nodos del centro) Se deviden los nodos del centro de i en i y cada grupo de i se conecta con un nodo distinto de la derecha.
Si un grupo no está completo no se considera. El siguiente debe de cubrir los nodos que no están cubiertos en este.

Podemos elegir los rojos.

Con los verdes sería suficiente

Razón $13/6 > 2$.Puede llegar a $\log(n)$.

n número de nodos centrales

Aproximación para Cubrimiento por Vértices

Algoritmo 2-Aproximado:

- $C = \emptyset$
- Mientras haya aristas en G
  - Elegir una arista cualquiera de G
  - Añadir sus dos extremos a C
  - Borrar de G los dos nodos y todas sus aristas

Este es un algoritmo 2-aproximado : todas las aristas tienen que tener un extremos en C. De esta forma, nunca elegimos más del doble de lo necesario.
Es el mejor algoritmo que se conoce. No se puede aproximar con $\delta = 1.1659$.

El problema del Corte Máximo: CM

Datos: Un grafo no dirigido $G = (V, E)$.
Problema: Partir V en dos conjuntos S y $V \setminus S$ de tal manera que haya un número máximo de arcos entre S y $V \setminus S$.

El problema de decisión es NP-completo.

El problema del Corte Máximo: CM

Datos: Un grafo no dirigido $G = (V, E)$.
Problema: Partir V en dos conjuntos S y $V \setminus S$ de tal manera que haya un número máximo de arcos entre S y $V \setminus S$.

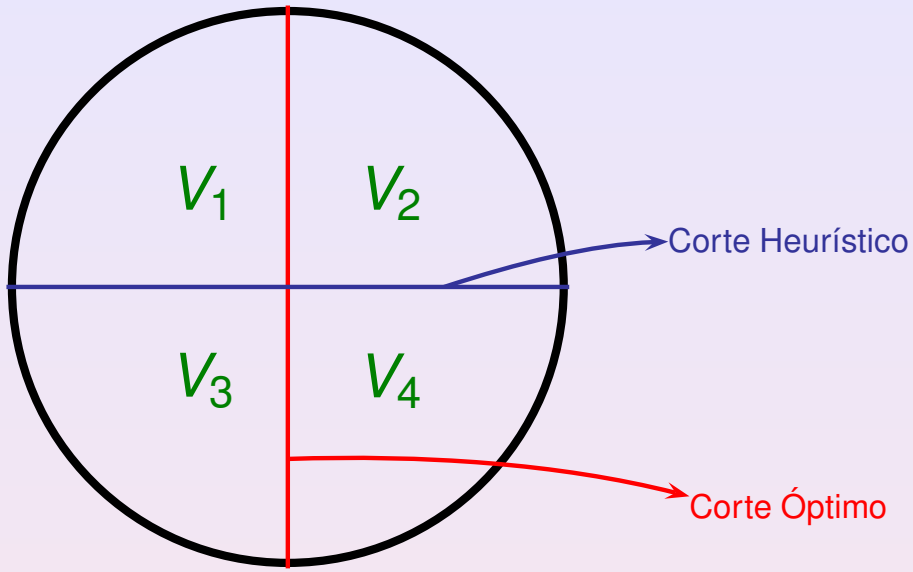
El problema de decisión es NP-completo.

Un algoritmo *Greedy*

Suponemos que no existen arcos que unen un nodo con el mismo: estos arcos se pueden quitar ya que no están nunca en el corte.
Si v es un nodo y A un subconjunto de nodos, consideremos $arc(x, A)$: el número de arcos que unen v con A.

- Comenzamos con un conjunto S arbitrario
- Mientras S cambie
  - Para cada vertice v del grafo
    - Si el vertice esta en S
      - Si $arc(v, V \setminus S) < arc(v, S)$
      - Eliminar v de S
    - Si el vertice no esta en S
      - Si $arc(v, V \setminus S) > arc(v, S)$
      - Añadir v a S

Es un algoritmo con razón de aproximación 2



Sea e_{ij}, $1 \leq i \leq j \leq 4$ el número de arcos entre V_i y V_j.
Para cada nodo de V_1 los arcos que van a nodos de V_1 y V_2 son menos que los que van a V_3 y V_4:
$\forall x_1 \in V_1, arc(x_1, V_1) + arc(x_1, V_2) \leq arc(x_1, V_3) + arc(x_1, V_4)$.
Sumando la desigualdad anterior en todos los nodos de V_1 obtenemos: $2e_{11} + e_{12} \leq e_{13} + e_{14}$
y, por tanto, $e_{12} \leq e_{13} + e_{14}$.

Repitiendo lo anterior para V_1, V_2, V_3, V_4, obtenemos

$$e_{12} \leq e_{13} + e_{14}, \quad e_{12} \leq e_{23} + e_{24},$$
$$e_{34} \leq e_{23} + e_{13}, \quad e_{34} \leq e_{14} + e_{24}$$

Sumando y dividiendo por dos se obtiene

$$e_{12} + e_{34} \leq e_{14} + e_{23} + e_{13} + e_{24}$$

También es obvio que $e_{14} + e_{23} \leq e_{14} + e_{23} + e_{13} + e_{24}$.
Sumando, las dos últimas desigualdades obtenemos:

$$e_{12} + e_{34} + e_{14} + e_{23} \leq 2.(e_{14} + e_{23} + e_{13} + e_{24})$$

Lo que nos disce que nuestro algoritmo heurístico, al menos, obtiene la mitad de arcos del óptimo: La razón de aproximación es 2.

Satisfacción Máxima: K-GMAXSAT

DATOS: Tenemos n variables $p_1, \ldots, p_n$ y m fórmulas booleanas: $\{\phi_1, \ldots, \phi_m\}$ tal que cada una de ellas tiene, a lo más, K variables.

Problema: Queremos encontrar una asignación de valores de verdad que maximice el número de fórmulas ciertas.

Algoritmo Aproximado

- Para una asignación de valores de verdad al azar, calculamos la p robabilidad de que ϕ_i sea cierta:

$$P(\phi_i) = \frac{t_i}{2^k}$$

donde t_i es el número de asignaciones a las variables que hacen que ϕ_i sea cierta y k el número de variables de la fórmula.
Esto se puede calcular ya que está limitado el número de variables en una fórmula.

- Para una asignación al azar de valores de verdad el número esperado de fórmulas que se satisfacen es

$$P(\Phi) = \sum_{i=1}^{m} P(\phi_i)$$

Algoritmo Aproximado

- Seleccionamos una variable x_1 cualquiera
- Si seleccionamos una variable x_1, tenemos que

$$P(\Phi) = 1/2\,(P(\Phi[x_1 = \mathit{verdadero}]) + P(\Phi[x_1 = \mathit{falso}]))$$

donde $\Phi[x_1 = \mathit{verdadero}]$, $\Phi[x_1 = \mathit{falso}]$ son los conjuntos de fórmulas que se obtienen a partir de las originales, substituyendo x_1 por verdadero ($\neg x_1$ falso) y x_1 por falso ($\neg x_1$ verdadero) respectivamente.

Elegimos $x_1 = \mathit{verdadero}$ si $P(\Phi[x_1 = \mathit{verdadero}]) \geq P(\Phi[x_1 = \mathit{falso}])$ y $x_1 = \mathit{falso}$ en caso contrario.

- Si x_1 es verdadero repetimos lo mismo para el resto de las variables con $\Phi[x_1 = \mathit{verdadero}]$ y si lo hacemos falso trabajamos con $\Phi[x_1 = \mathit{falso}]$.

Ejemplo

Num. Esp. = 1+3/4+1+1/2+3/4+7/8+1/2+1=6+3/8

V, $r \vee q$, V, s
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, V

p Verd.

$p \vee \neg q$, $\neg p \vee r \vee q$, p, $\neg p \vee s$
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, $r \vee p \vee s$

Hacemos p verdadero

p Fals.

Num. Esp. = 3/4+7/8+1/2+3/4+3/4+7/8+1/2+7/8

= 5+7/8

$\neg q$, V, F, V
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, $r \vee s$

Num. Esp. = 1/2+1+0+1+3/4+7/8+1/2+3/4= 5+3/8

Ejemplo

Num. Esp. = 1+3/4+1+1/2+3/4+7/8+1/2+1=6+3/8

V, $r \vee q$, V, s
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, V

p Verd.

$p \vee \neg q$, $\neg p \vee r \vee q$, p, $\neg p \vee s$
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, $r \vee p \vee s$

Hacemos p verdadero

p Fals.

Num. Esp. = 3/4+7/8+1/2+3/4+3/4+7/8+1/2+7/8

= 5+7/8

$\neg q$, V, F, V
$q \vee \neg s$, $\neg r \vee s \vee \neg q$, $\neg s$, $r \vee s$

Num. Esp. = 1/2+1+0+1+3/4+7/8+1/2+3/4= 5+3/8

Propiedades del Algoritmo

- De esta manera, el valor esperado del fórmulas que se satisfacen nunca decrece.
- Cuando llegamos al final (todas las variables tienen su valor de verdad), el numero esperado es el número exacto de fórmulas que se satisfacen, que es al menos el número original $P(\Phi)$.
- El óptimo del problema es menor o igual que el número de fórmulas ϕ_i para las que $P(\phi_i) > 0$, lo que denotamos como $\#\{\phi_i : P(\phi_i) > 0\}$.

Razón de aproximación

La razón de aproximación es:

$$\frac{OPT(x)}{APROX(x)} \leq \frac{\#\{\phi_i \,:\, P(\phi_i) > 0\}}{APROX(x)} \leq$$

$$\frac{\#\{\phi_i \,:\, P(\phi_i) > 0\}}{P(\phi_1) + \ldots + P(\phi_m)} \leq \frac{1}{\min\{P(\phi_i) \,:\, P(\phi_i) > 0\}}$$

Este es un algoritmo δ-aproximado con $\delta = \frac{1}{\min\{P(\phi_i) : P(\phi_i) > 0\}}$.

- Para fórmulas con k variables como máximo es δ-aproximado $\delta = 2^k$.
- Para cláusulas es un algoritmo con razón 2.
- Para cláusulas con k literales distintos es $\delta = \frac{1}{1-(2^{-k})}$.

Resultado

Si P$\neq$ NP el problema del viajante de comercion no tiene un algoritmo con umbral δ polinómico.

Supongamos que existe un algoritmo δ-aproximado polinómico para el problema del viajante de comercio ($\delta < \infty$). Entonces, construiremos el siguiente algoritmo para el circuito hamiltoniano.
Sea $G = (V, E)$ un grafo, entonces construimos un problema del viajante de comercio con los mismos nodos y en el que

$$d(i,j) = \begin{cases} 1 & \text{si } (i,j) \in E \\ |V|.\delta & \text{si } (i,j) \notin E \end{cases}$$

Aplicamos el algoritmo polinómico con razón δ.

Si el algoritmo encuentra un circuito de coste $|V|$, existe un circuito hamiltoniano en G.
Si el algoritmo aproximado no encuentra un circuito de coste $|V|$, devolverá un circuito de coste mayor de $|V|.\delta$, entonces como $\frac{\text{Coste Circuito}}{\text{Optimo}} \leq \delta$
deducimos

$$\text{Optimo} \geq \frac{\text{Coste Circuito}}{\delta} > |V|$$

Como consecuencia, Optimo $> |V|$ y el óptimo tiene un arco que no es de G y, por tanto, G no tiene un circuito hamiltoniano.

El problema de la Mochila

Pesos: $w_1, \ldots, w_n$ **Valores:** $v_1, \ldots, v_n$
Peso límite: W
Problema: Encontrar $S \subseteq \{1, \ldots, n\}$ tal que $\sum_{i \in S} w_i \leq W$ y $\sum_{i \in S} v_i$ sea máximo.

Podemos suponer que todo objeto tiene un peso $w_i \leq W$ (todos los objetos caben en la mochila). Si un objeto no cabe, entonces se quita del problema y la solución es la misma.

El problema de la Mochila

Pesos: $w_1, \ldots, w_n$ **Valores:** $v_1, \ldots, v_n$
Peso límite: W
Problema: Encontrar $S \subseteq \{1, \ldots, n\}$ tal que $\sum_{i \in S} w_i \leq W$ y $\sum_{i \in S} v_i$ sea máximo.

Podemos suponer que todo objeto tiene un peso $w_i \leq W$ (todos los objetos caben en la mochila). Si un objeto no cabe, entonces se quita del problema y la solución es la misma.

Algoritmo Pseudo-Polinómico

Sea $V = \max\{v_1, \ldots, v_n\}$
Para $i = 0, 1, \ldots, n$ y $0 \leq v \leq nV$ calculamos:
$W(i, v)$: Mínimo peso que se puede conseguir eligiendo items entre los i primeros de valor v exactamente.
Se calcula con:
$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}$

Algoritmo Pseudo-Polinómico

Para $i = 0, 1, \ldots, n$ y $0 \leq v \leq nV$ calculamos:
$W(i, v)$: Mínimo peso que se puede conseguir eligiendo items entre los i primeros de valor v exactamente.
Se calcula con:
$W(i+1, v) = \min\{W(i, v), W(i, v - v_{i+1}) + w_{i+1}\}$
Complejidad $O(n^2 V)$. Es Pseudo-polinómico (exponencial en función de la longitud de V).
Cuando se ha calculado $W(n, v)$ es fácil calcular el óptimo:
Comparamos, desde $i = nV$ hasta $i = 0$ los valores $W(n, i)$ con el límite W. El primer i para el que $W(n, i) \leq W$ es el óptimo del problema.

Algoritmo Aproximado

Dado un ejemplo $I = (w_1, \ldots, w_n, W, v_1, \ldots, v_n)$ consideramos

$$I' = (w_i, \ldots, w_n, W, v'_1, \ldots, v'_n)$$

donde $v'_i = 2^b \left[\frac{v_i}{2^b}\right]$.
(Los b bits menos significativos se reemplazan por 0).
Entonces aplicamos el algoritmo pseudo-polinómico, pero quitando los 0s a la derecha de los números y añadiéndolos al resultado.
Este tendrá una complejidad en este caso de $O\left(\frac{n^2 V}{2^b}\right)$.

Algoritmo Aproximado

Se obtiene un algoritmo δ-aproximado, haciendo $b = \left\lceil \log(\frac{(\delta-1).V}{n}) \right\rceil$.

Sea S el conjunto óptimo del problema original y S' el que obtenemos en el algoritmo aproximado. Vamos a llamar a $V(S) = \sum_{i \in S} v_i$ el valor conseguido con el algoritmo óptimo y $V(S') = \sum_{i \in S} v_i$ el valor conseguido con el algoritmo aproximado.

Supondremos que $V(S') \geq V$ (como todos los objetos caben, siempre podemos considerar la solución que mete el objeto de mayor valor).

Tenemos que:

$V(S) = \sum_{i \in S} v_i \geq V(S') = \sum_{i \in S'} v_i \geq \sum_{i \in S'} v'_i \geq \sum_{i \in S} v'_i \geq \sum_{i \in S}(v_i - 2^b) \geq (\sum_{i \in S} v_i) - n.2^b = V(S) - n.2^b$

$$V(S) \geq V(S') \geq V(S) - n.2^b$$

Por tanto, la diferencia entre el óptimo y lo que calculamos es menor o igual a $n.2^b$.

Si $b = \left\lfloor \log(\frac{(\delta-1).V}{n}) \right\rfloor$,y si el valor de nuestro algoritmo aproximado es, al menos, V (esto ocurre si todos los items caben en la mochila) tenemos que

$\frac{V(S)}{V(S')} \leq \frac{V(S')+n2^b}{V(S')} \leq \frac{V(S')+n\frac{(\delta-1)V}{n}}{V(S')} \leq \frac{V(S')+n\frac{(\delta-1)V(S')}{n}}{V(S')} =$
$1 + \frac{V.(\delta-1)}{V} = \delta$

Es decir es un algoritmo δ-aproximado.

La complejidad es $O\left(\frac{n^2 V}{2^b}\right)$, que para $b = \left\lfloor \log(\frac{(\delta-1).V}{n}) \right\rfloor$, se convierte en $O(n^3/(\delta-1))$.

Esquema de Aproximación Polinómico

Un problema de optimización Π tiene un esquema de aproximación polinómico si existe un algoritmo que para cada $\delta > 1$ y cada ejemplo x de Π, devuelve una aproximación de grado δ del óptimo de x, en tiempo polinómico en función de $|x|$ (el polinomio puede depender de δ).
Si la dependencia de δ se puede expresar como un polinomio en $(1/(\delta - 1))$, se dice que es un esquema de aproximación polinómico total .

El problema de la mochila tiene un esquema de aproximación polinómico total

Conjunto Independiente

Teorema

Si existe un algoritmo δ_0-aproximado polinómico con $\delta_0 < \infty$ para el máximo conjunto independiente, entonces existe un esquema de aproximación polinómico.

La clase **NPO**

Definición **NPO**

Un problema de optimización está en **NPO** si y solo si se cumplen las siguientes condiciones:

1. El conjunto de entradas correctas se puede reconocer en tiempo polinómico.
2. Existe un polinomio q tal que para cada caso del problema x, y para cada solución factible y, tenemos que $|y| \leq q(|x|)$, y además para cada $|y| \leq q(|x|)$ es decidible en tiempo polinómico si es una solución factible del problema.
3. La función de costo se puede calcular en tiempo polinómico.

Ejemplo: El cubrimiento mínimo por vértices.

La clase **PO**

Un problema de optimización está en la clase **PO** si está en **NPO** y existe un algoritmo polinómico tal que para cada caso del problema x existe un algoritmo polinómica que calcula la solución óptima y su coste.

Ejemplo: La distancia mínima en grafos.

Optimización $\rightarrow$ Decisión

Dado un problema de optimización (minimización o maximización), siempre podemos asociarle un problema de decisión: por ejemplo para un problema de minización, se da una cota K y se pregunta si existe una solución factible de costo menor o igual a K.

Teorema

Para cualquier problema de optimización de NPO su correspondiente problema de decisión está en NP.

Teorema

Si $P \neq NP$ entonces $PO \neq NPO$.

La clase **APX**

Definición: Clase APX

La clase **APX** es la clase de todos los problemas tales que admiten un algoritmo δ-aproximado polinómico para $\delta < \infty$.

Ejemplo: *Problemas en* APX

Cubrimiento mínimo por vértices, problema de la mochila, corte máximo, máximo número de cláusulas satisfechas, etc...

Ejemplo: *Problemas que no están en* APX

Problema del viajante de comercio, máximo clique, máximo conjunto independiente, etc...

Teorema

Si $\mathbf{N} \neq \mathbf{NP}$, entonces $\mathbf{APX} \neq \mathbf{NPO}$

La clase **APX**

Definición: Clase APX

La clase **APX** es la clase de todos los problemas tales que admiten un algoritmo δ-aproximado polinómico para $\delta < \infty$.

Ejemplo: *Problemas en* APX

Cubrimiento mínimo por vértices, problema de la mochila, corte máximo, máximo número de cláusulas satisfechas, etc...

Ejemplo: *Problemas que no están en* APX

Problema del viajante de comercio, máximo clique, máximo conjunto independiente, etc...

Teorema

Si $\mathbf{N} \neq \mathbf{NP}$, entonces $\mathbf{APX} \neq \mathbf{NPO}$

La clase **APX**

Definición: Clase APX

La clase **APX** es la clase de todos los problemas tales que admiten un algoritmo δ-aproximado polinómico para $\delta < \infty$.

Ejemplo: *Problemas en* APX

Cubrimiento mínimo por vértices, problema de la mochila, corte máximo, máximo número de cláusulas satisfechas, etc...

Ejemplo: *Problemas que no están en* APX

Problema del viajante de comercio, máximo clique, máximo conjunto independiente, etc...

Teorema

Si $\mathbf{N} \neq \mathbf{NP}$, entonces $\mathbf{APX} \neq \mathbf{NPO}$

La clase **APX**

Definición: Clase APX

La clase **APX** es la clase de todos los problemas tales que admiten un algoritmo δ-aproximado polinómico para $\delta < \infty$.

Ejemplo: *Problemas en* APX

Cubrimiento mínimo por vértices, problema de la mochila, corte máximo, máximo número de cláusulas satisfechas, etc...

Ejemplo: *Problemas que no están en* APX

Problema del viajante de comercio, máximo clique, máximo conjunto independiente, etc...

Teorema

Si **N** $\neq$ **NP**, entonces **APX** $\neq$ **NPO**

La clase PTAS

Definición de **PTAS**

Es la clase de problemas con un esquema de aproximación polinómico.

Ejemplo: Un problema de **PTAS**

el problema de la mochila en el que puede haber tantas copias como se quiera de cada objeto.

Ejemplo: Problemas de **APX** que no está en **PTAS** (si **P**$\neq$ **NP**)

El corte máximo o el cubrimiento mínimo.

Teorema

En grafos planares, el cubrimiento mínimo está en **PTAS**.

La Clase **FPTAS**

Definición: Problemas con un esquema de aproximación polinómico total (FPTAS)

Para cada ejemplo x del problema Π, y todo número racional $\delta > 1$, el algoritmo devuelve para la entrada (x, δ) una solución δ-aproximada que es polinómico en $|x|$ y $(1/(\delta - 1))$.

Definición: Problemas acotados polinómicamente

Un problema de optimización está acotado polinómicamente si y solo si existe un polinomio p tal que para todo ejemplo x y para toda solución factible y de x, entonces

$$c(x, y) \leq p(|x|)$$

donde $c(x, y)$ es el coste de la solución y.

Ejemplo

El problema del viajante de comercio no está acotado polinómicamente. El del cubrimiento mínimo por vértices si lo está.

FPTAS y PTAS

Teorema

No existe un problema NP-difícil acotado polinómicamente con un esquema de aproximación polinómico total.

Corolario

Si P $\neq$ NP entonces PTAS $\neq$ FPTAS

Ejemplo

El máximo conjunto independiente para grafos planares estaría en PTAS $\setminus$ FPTAS.

Problemas Pseudo-polinómicos

- En el problema del corte máximo la dificultad está en su estructural combinatoria: no hay números, sólo un grafo con sus vértices y arcos.
- El problema de la mochila puede resolverse por programación dinámica en tiempo $O(n^2 p_{max})$ donde p_{max} es el entero de tamaño máximo que aparece en el problema. Eso no implica que el problema sea polinómico, pero su complejidad se debe al tamaño de los números que aparecen.

Problemas Pseudo-polinómicos

Los problemas que tienen un algoritmo tal que para cada ejemplo del problema x encuentra la respuesta correcta en un tiempo que depende polinómicamente de $|x|$ y del entero mayor que aparezca en la especificación de x se dice que son Pseudo-Polinómicos.

Problemas NP-completos y pseudo-polin.

Teorema

Si Π es un problema que está en **FPTAS** , y si existe un polinomio p tal que para cada ejemplo x, el coste óptimo del problema $c^*(x)$ verifica:

$$c^*(x) \leq p(|x|, \max(x))$$

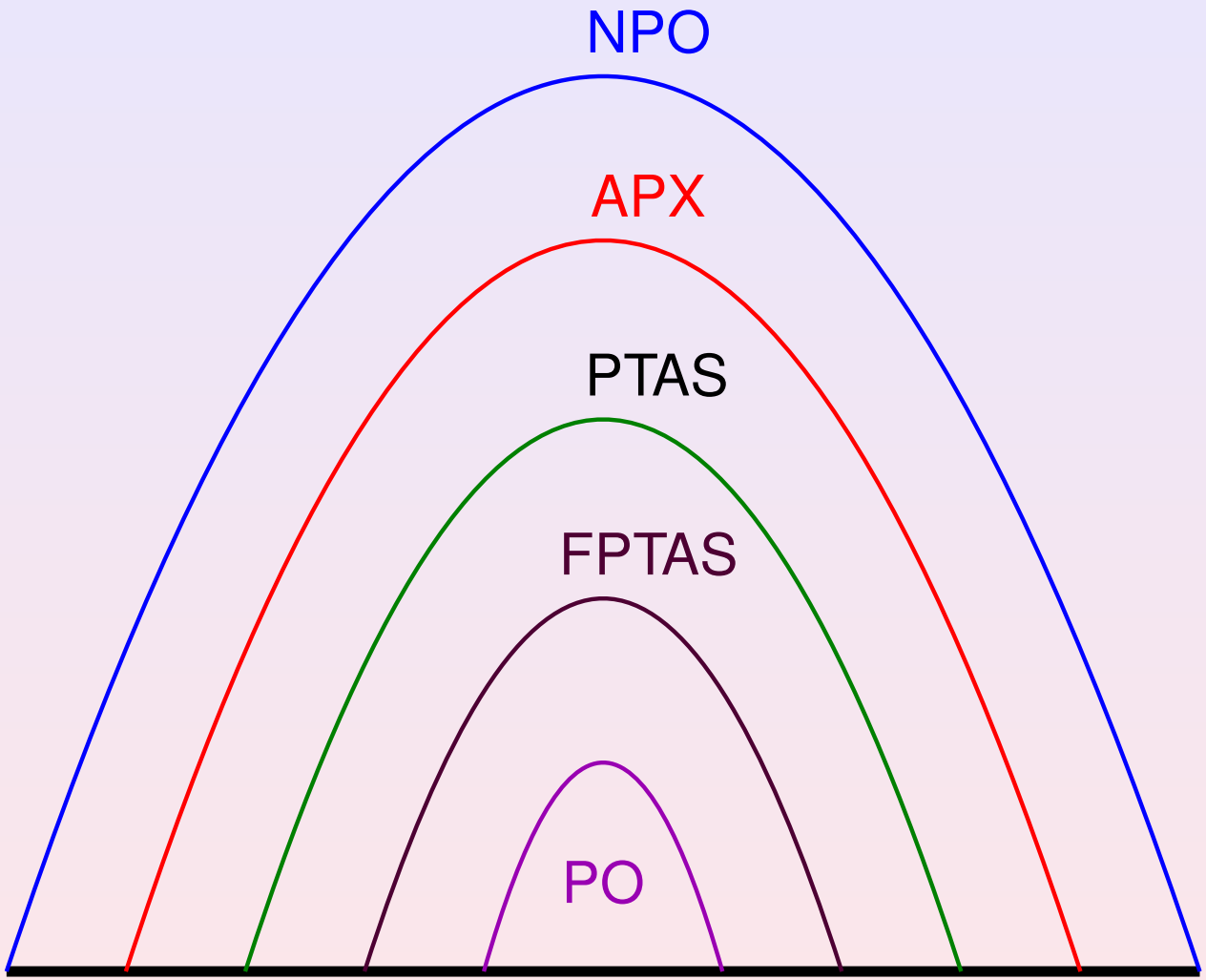
donde $\max(x)$ es el entero mayor que aparece en x, entonces el problema es pseudo-polinómico .

Demostración

Basta con llamar al esquema de aproximación polinómico total para el problema x y para la aproximación

$$\delta = \frac{p(|x|, \max(x)) + 2}{p(|x|, \max(x)) + 1}$$

Clases de Aproximación



Problemas Completos y Difíciles

Existe un concepto de reducción (L-reducción para problemas NPO).

Definición

Dada una clase **C** de problemas en **NPO**, decimos que un problema Π es **C**-difícil si y solo si cualquier otro problema de **C** es L-reducible a él. Y es **C**-completo si, además, está en **C**.

Teorema

Si un problema **NPO**-completo tiene un esquema de aproximación polinómico, entonces todos los problemas de **NPO** también lo tienen.

Un teorema análogo se puede dar para la clase **APX**.

Ejemplos

Ejemplo: Problemas NPO-completos

Máxima, mínima satisfacción ponderada, máxima y mínima programación lineal $\{0, 1\}$ y el problema del viajante de comercio.

Ejemplo: Problemas APX-completos

Satisfacción máxima de cláusulas de longitud 3, de cláusulas de longitud 2, el corte máximo, viajante de comercio con desigualdad triangular, mínimo cubrimiento por vértices, mínimo conjunto de ruptura de ciclos en grafos dirigidos.