

# Algoritmos aproximados

Elvira Mayordomo

Universidad de Zaragoza

27 de noviembre de 2018

- 1 **Introducción**
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

Este tema está basado en:

- El capítulo 6 (6.8) de Steven S. Skiena. The Algorithm Design Manual. Springer 2008.
- El capítulo 35 de T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein. Introduction to Algorithms. The MIT Press 2009.
- El capítulo 13 de G. Brassard, P. Bratley. Fundamentos de Algoritmia. Prentice Hall 1997.

# Motivación: Salvando los NP-difíciles

- Tiempo exponencial puede ser aceptable para las pequeñas entradas (Fuerza bruta).
- A veces se pueden aislar los casos especiales que se pueden ejecutar en tiempo polinomial (técnicas de AB).
- Soluciones casi óptimas pueden ser aceptables (algoritmos de aproximación).
- Veremos como algunos NP-difíciles tienen algoritmos de aproximación muy buenos mientras que para otros aproximar mínimamente es tan difícil como resolver el problema.

# Contenido de este tema

- 1 Introducción
- 2 **Problemas de optimización**
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

- En este tema nos centraremos en resolver *problemas de optimización*.
- Nos conformaremos con soluciones aproximadas.

- Se trata de:
  - ▶ Buscar la solución (camino, etc) más grande/larga, etc que cumpla ...
  - ▶ Buscar la solución (camino, etc) más pequeña/corta, etc que cumpla ...
- En general tenemos:
  - ▶ Para cada entrada un espacio de **soluciones candidatas** (por ejemplo, caminos)
  - ▶ Una medida de bondad de la solución, **coste** o función objetivo (por ejemplo, la longitud del camino)
- **Optimizar** = buscar la mejor solución o solución óptima

# Ratios de aproximación

- $c(x)$  es el coste de la solución óptima
- $\hat{c}(x)$  es el coste de la solución producida por el algoritmo de aproximación
- Un algoritmo tiene una ratio de aproximación de  $\rho(n)$  si para una entrada  $x$  de tamaño  $n$ ,  $\hat{c}(x)$  está dentro de un factor de  $\rho(n)$  de  $c(x)$



- Problema de **maximización**

- ▶  $0 < \hat{c}(x) \leq c(x)$
- ▶  $c(x)/\hat{c}(x)$  factor por el cual el coste de la solución óptima es mayor que el coste de la solución aproximada

$$\rho(n) = \max_{|x|=n} \frac{c(x)}{\hat{c}(x)}$$

- Problema de **minimización**

- ▶  $0 < c(x) \leq \hat{c}(x)$
- ▶  $\hat{c}(x)/c(x)$  factor por el cual el coste de la solución aproximada es mayor que el coste de la solución óptima

$$\rho(n) = \max_{|x|=n} \frac{\hat{c}(x)}{c(x)}$$

# ¿Cómo funcionan los algoritmos de optimización

- Explotan la naturaleza del problema
- Usan técnicas voraces
- Usan programación lineal
- Usan programación dinámica
- ...

- $c(x)$  es el coste de la solución óptima (**no sabemos cómo conseguirlo**)
- $\hat{c}(x)$  es el **coste conseguido por el algoritmo** de aproximación
- $\rho(n)$  es el máximo (para  $|x| = n$ ) del ratio entre  $c(x)$  y  $\hat{c}(x)$  (en el orden adecuado, siempre  $\leq 1$ )
- Cuanto más cerca de 1 está  $\rho(n)$  mejor es la aproximación (con  $\rho(n) = 1$  tenemos aproximación perfecta)

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 **Diferentes tipos de aproximación**
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

# Diferentes tipos de aproximación

(Supongamos que se trata de un problema de maximización)

- Ratio de aproximación logarítmico (**log-aproximable**)

$$c(x)/\hat{c}(x) \leq \rho(n) = O(\log n)$$

- Ratio de aproximación  $B$  (constante) ( **$B$ -aproximable**)

$$c(x)/\hat{c}(x) \leq \rho(n) \leq B$$

- Ratio de aproximación asintóticamente pequeño ( **$\epsilon$ -aproximable**)

$$\forall \epsilon, c(x)/\hat{c}(x) \leq \rho(n) \leq (1 + \epsilon)$$

- **FPTAS** (“Full polynomial time approximation scheme”): el algoritmo de aproximación tiene ratio de aproximación asintóticamente pequeño y el tiempo para ratio  $\epsilon$  y entradas de tamaño  $n$  es polinómico en  $n$  y  $1/\epsilon$

# Diferentes tipos de aproximación

- Para cualquiera de los tipos de aproximación anteriores, el problema puede ser no aproximable (si  $P \neq NP$ ), por ejemplo  
Para cualquier  $B$ , TSP no es  $B$  aproximable (si  $P \neq NP$ )  
es decir, para cualquier  $B$ ,  $B$ -aproximar TSP es tan difícil como resolverlo

- Los diferentes tipos de aproximación dependen de lo grande que pueda ser  $\rho(n)$  (cuanto más pequeño  $\rho(n)$  mejor aproximación)
- Nos interesarán los tres casos de  $\rho(n)$  constante, logarítmico, asintóticamente cercano a 1
- Veremos muchos algoritmos de aproximación y algún resultado negativo (no existen algoritmos que aproximen con ratio  $x$ )

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 **Cobertura de Vértices**
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen



# Cobertura de Vértices

*Problema:* Cobertura de Vértices

*Entrada:* Un grafo  $G$  (con vértices  $V$ ) y  $k \in \mathbb{N}$ .

*Salida:* ¿Existe un conjunto  $U$  de  $k$  vértices de  $G$  tal que cada arista  $(i, j)$  de  $G$  cumple que  $i \in U$  ó  $j \in U$ ?

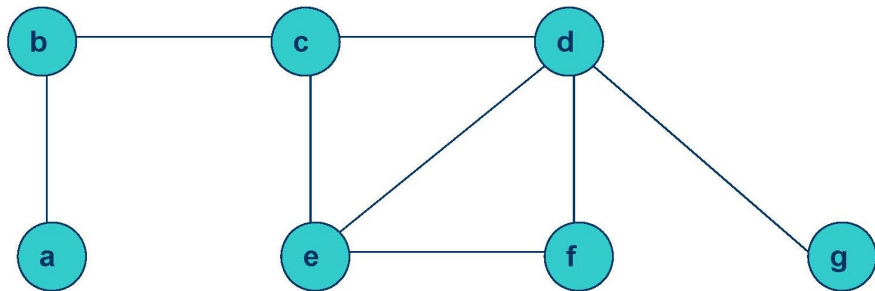
- Cobertura de Vértices es NP-difícil
- Vamos a considerar la versión de optimización, encontrar un cubrimiento óptimo

*Problema:* VC

*Entrada:* Un grafo  $G$  (con vértices  $V$ ).

*Salida:* Encontrar un conjunto lo menor posible  $U$  de vértices de  $G$  tal que cada arista  $(i, j)$  de  $G$  cumple que  $i \in U$  ó  $j \in U$

## Considerar el grafo ...



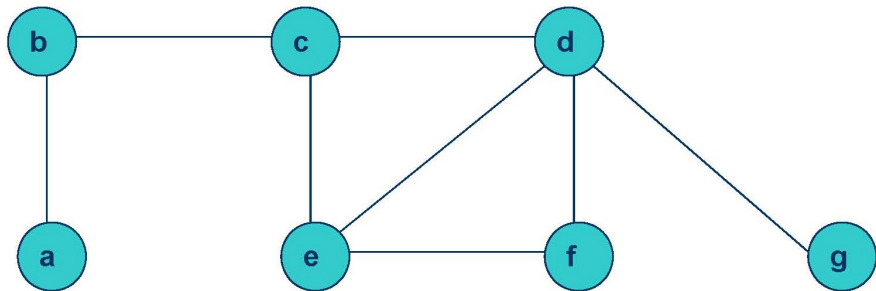
- Por inspección, el cubrimiento óptimo es  $\{b, d, e\}$
- $c(G) = \text{tamaño de la solución óptima} = 3$

# Algoritmo de aproximación de VC

APROXVC( $G$ )

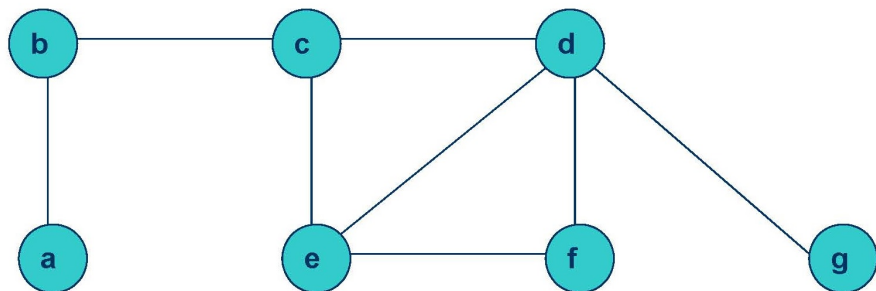
- 1  $U = \emptyset$
- 2  $E' =$ aristas de  $G$
- 3 **while**  $E' \neq \emptyset$
- 4     sea  $(u, v) \in E'$
- 5      $U = U \cup \{u, v\}$
- 6     Borrar todas las aristas de  $E'$  que tienen  $u$  ó  $v$
- 7 Resultado  $U$

## De vuelta a nuestro grafo ...



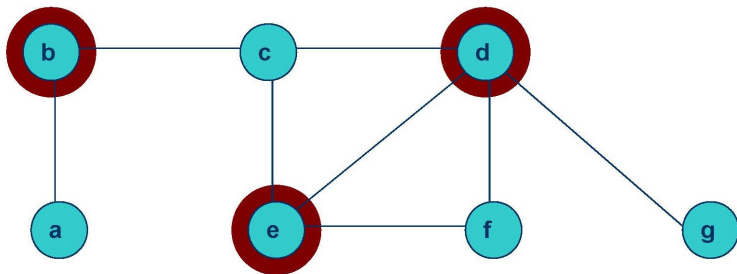
- $E' = \{(a, b)(b, c)(c, d)(c, e)(d, e)(d, f)(d, g)(e, f)\}$
- Cubrimiento aproximado  $\hat{c}(G) = 6$

## Con más suerte ...



- $E' = \{(d, e)(c, d)(c, e)(d, f)(d, g)(e, f)(b, c)(a, b)\}$
- Cubrimiento aproximado  $\hat{c}(G) = 4$

## ¿Es posible acercarse más a la solución óptima?



- En este caso y con nuestro algoritmo, ¡NO!

- Tiempo  $O(n + m)$  ( $n$  vértices y  $m$  aristas)
- 2-aproximación: vamos a verlo (cierto para el ejemplo anterior)  
 $\hat{c}(G) = 6$ ,  $c(G) = 3$ )

## 2-aproximable

- $A$  = conjunto de aristas elegidas por el algoritmo AproxVC
- No hay dos aristas de  $A$  con un punto en común, así que no hay dos aristas de  $A$  cubiertas por el mismo vértice de un cubrimiento
  - ▶ cota inferior  $c(G) \geq |A|$
- Elegimos una arista para la que ninguno de los dos extremos está ya en  $U$ 
  - ▶ cota superior  $\hat{c}(G) = 2|A|$
- Por tanto  $2c(G) \geq 2|A| = \hat{c}(G)$
- Luego  $\hat{c}(x)/c(x) \leq 2$



- Hemos visto que podemos aproximarlo con un ratio 2
- No se puede aproximar con ratio 1,1666 (si  $P \neq NP$ )

# Qué hemos aprendido

- Aunque el algoritmo es simple, no es estúpido **Por ejemplo, considera la heurística de seleccionar un solo vértice en lugar de los 2 y una estrella ...**
- Voraz no es siempre la respuesta **Quizás la heurística más natural es seleccionar el vértice de mayor grado ... sin embargo con casi empates puede ir realmente mal y ser  $\Theta(\log n)$  aproximado**
- Hacer una heurística más complicada no la hace necesariamente mejor **Por ejemplo podríamos completar el algoritmo anterior seleccionando la arista con vértices de mayor grado ... pero eso no mejora el caso peor y lo hace más difícil de analizar**
- Un paso de limpieza a posteriori no es malo **Por ejemplo quitar los vértices innecesarios del resultado puede mejorar el resultado, aunque no el caso peor**
- Recuerda que en los algoritmos de aproximación hay que considerar el ratio de aproximación en el **caso peor**, no pensar en casos concretos en los que funcione bien

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 **TSP métrico**
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

*Problema:* TSP

*Entrada:*  $n$  el número de ciudades, la matriz de distancias  $n \times n$  y cota superior  $k$ .

*Salida:* ¿Existe un recorrido por las  $n$  ciudades, sin repeticiones y volviendo al punto de partida con distancia total  $\leq k$ ?

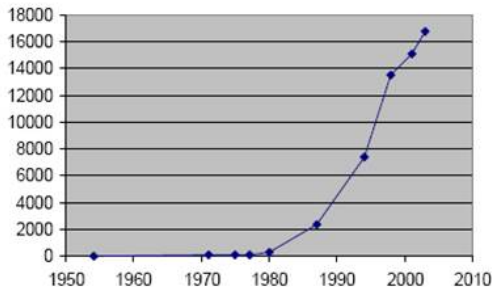
- TSP es NP-difícil
- Vamos a considerar la versión de optimización, encontrar un recorrido óptimo

*Problema:* TSP minimización

*Entrada:*  $n$  el número de ciudades y  $M$  la matriz de distancias  $n \times n$ .

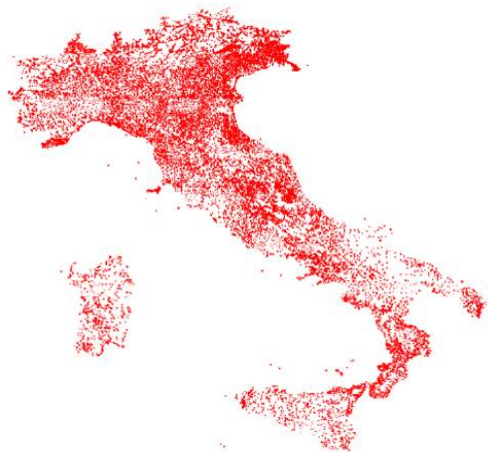
*Salida:* Encontrar un recorrido por las  $n$  ciudades, sin repeticiones y volviendo al punto de partida con distancia total la mínima posible

- Prestaciones de los **métodos exactos** para resolver el problema
- Año vs tamaño del problema resuelto óptimamente

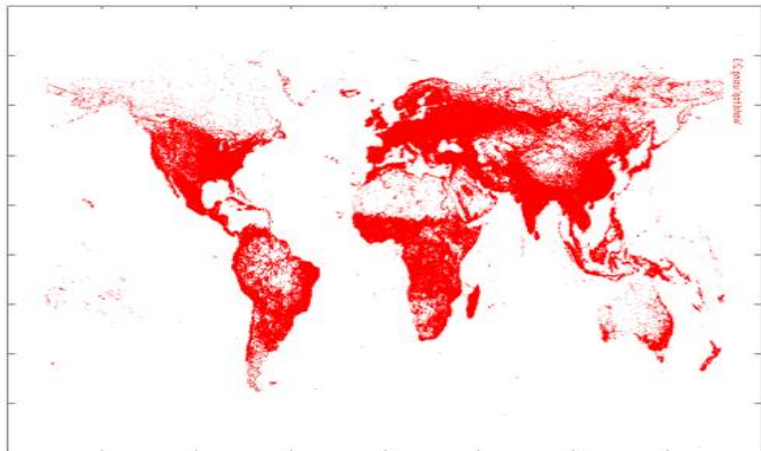


- El mayor problema resuelto óptimamente hasta 2006 es una entrada de 85.900 ciudades
- Usando heurísticas, varias entradas de millones de ciudades se han resuelto dentro del 1 % de la solución óptima
- Más información en <http://www.math.uwaterloo.ca/tsp/optimal/index.html>

# Sobre TSP ...: solución óptima a ...



# Sobre TSP ...: solución heurística a ...





- La distancia satisface la desigualdad triangular para todas las ciudades  $u, v, w$

$$d(u, w) \leq d(u, v) + d(v, w)$$

- **TSP métrico sigue siendo NP-difícil**
- Parece una restricción trivial ... ¿o no?
  - ▶ Piensa en la distancia definida como tiempo de vuelo entre las dos ciudades **No satisface la desigualdad triangular**
  - ▶ O simplemente en el camino que selecciona un GPS con la opción "Ruta más rápida"
  - ▶ O el precio de volar de una ciudad a otra ...

- Grafos Eulerianos y circuitos Eulerianos:
  - ▶ **Ciclo Euleriano:** ciclo que usa cada arista exactamente una vez
  - ▶ **Grafo Euleriano:** grafo con un ciclo Euleriano
  - ▶ **Propiedad:** Un grafo es Euleriano si y sólo si cada vértice tiene grado par.

# “Spanning trees” o árboles de recubrimiento

- Un **árbol** es un grafo conexo sin ciclos
- Un **spanning tree** de un grafo es un árbol formado por todos los vértices y algunas aristas
- Un **spanning tree mínimo** es el que tiene menor distancia total

# Algoritmo de aproximación de TSP métrico

*Problema:* TSP métrico

*Entrada:*  $n$  el número de ciudades y  $M$  la matriz de distancias  $n \times n$  que cumplen la desigualdad triangular.

*Salida:* Encontrar un recorrido por las  $n$  ciudades y volviendo al punto de partida con distancia total la mínima posible

APROXMTSP( $n, M$ )

- 1  $G =$  grafo con distancias representado por  $M$
- 2 Encontrar  $T$  un spanning tree mínimo de  $G$
- 3 Doblar cada arista de  $T$  para obtener  $G'$  que es un grafo Euleriano
- 4 Encontrar  $E$  un circuito Euleriano de  $G'$
- 5  $F =$  los vértices de  $G$  en el orden que aparecen por primera vez en  $E$
- 6 Resultado:  $F$

- Encontrar  $T$  un spanning tree mínimo de  $G$ 
  - ▶ **Algoritmo de Kruskal** (Visto en AB: coste  $O(n^2 \log n)$ ):
  - ▶ Se basa en la propiedad de los árboles de recubrimiento de coste mínimo: Partiendo del árbol vacío, se selecciona en cada paso la arista de menor etiqueta que no provoque ciclo sin requerir ninguna otra condición sobre sus extremos.
- Encontrar  $E$  un circuito Euleriano de  $G'$ 
  - ▶ Se trata de atravesar  $T$  en profundidad ( $O(n^2)$ )
  - ▶ En general se puede encontrar un circuito Euleriano de cualquier grafo en tiempo  $O(m)$  ( $m$  es el número de aristas)
- **Tiempo total:**  $O(n^2 \log n + n^2) = O(n^2 \log n)$

# Algoritmo de aproximación de TSP métrico

APROXMTSP( $n, M$ )

- 1  $G =$  grafo con distancias representado por  $M$
- 2 Encontrar  $T$  un spanning tree mínimo de  $G$
- 3 Duplicar cada arista de  $T$  para obtener  $G'$  que es un grafo Euleriano
- 4 Encontrar  $E$  un circuito Euleriano de  $G'$
- 5  $F =$  los vértices de  $G$  en el orden que aparecen por primera vez en  $E$
- 6 Resultado:  $F$

- Es una 2-aproximación:
  - ▶ El resultado es  $F$  es quitar vértices repetidos a  $E$  circuito Euleriano ( $\hat{c}(n, M) = c(F)$ )
  - ▶  $\hat{c}(n, M) \leq c(E)$  Por la **desigualdad triangular**, nos quedamos con la primera aparición de cada vértice
  - ▶  $c(E) \leq 2c(T)$  (cada arista de  $T$  está duplicada)
  - ▶ **Cota inferior**  $\hat{c}(n, M) \leq c(E) \leq 2c(T)$
  - ▶ **Cota superior**  $c(T) \leq c(n, M)$  Porque si quitamos una arista de un recorrido se convierte en un árbol, y  $T$  es el árbol de coste mínimo
  - ▶  $\hat{c}(n, M) \leq 2c(T) \leq 2c(n, M)$
  - ▶ Luego  $\hat{c}(n, M)/c(n, M) \leq 2$

## ¿Podemos mejorar la aproximación?

- El punto crítico es  $c(E) \leq 2c(T)$
- Se debe a que hay que “Duplicar cada arista de  $T$  para obtener  $G'$  que es un grafo Euleriano”
- Si podemos evitar esta duplicación podemos mejorar la aproximación
- Existen algoritmos menos costosos para convertir  $T$  en un grafo Euleriano
- Se basan sólo en los vértices de grado impar y usan “perfect matching”
- Con esta idea se puede mejorar la aproximación a 1,5
- Es el famoso algoritmo de Christofides (1976)



- Tenemos una aproximación con ratio 1,5
- No se puede aproximar con ratio 1,013 (si  $P \neq NP$ )

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 **Set cover**
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

*Problema:* **Set Cover**

*Entrada:* Una colección  $S_1, \dots, S_m$  de  $m$  conjuntos,  $k \leq m$ .

*Salida:* ¿Existen  $i_1, \dots, i_k$  con

$$S_{i_1} \cup \dots \cup S_{i_k} = S_1 \cup \dots \cup S_m,$$

es decir, que la unión de los  $k$  sea la misma que la de la colección completa?

- Importante:  $k$  es el **número de subconjuntos**, no de elementos
- **Ejemplo:** Sean  $m$  expertos y  $S_1, \dots, S_m$  las destrezas que aporta cada uno de los expertos,  $k \leq m$ . ¿Podemos nombrar un comité de  $k$  expertos o menos que cubran todas las destrezas?

- Set Cover es NP-difícil
- Vamos a considerar la **versión de optimización**, encontrar una subcolección óptima

*Problema:* Opt-Set-Cover

*Entrada:* Una colección  $S_1, \dots, S_m$  de  $m$  conjuntos.

*Salida:* Encontrar  $i_1, \dots, i_k$  con  $k$  el menor posible y

$$S_{i_1} \cup \dots \cup S_{i_k} = S_1 \cup \dots \cup S_m$$

# Algoritmo de aproximación de Set cover (voraz)

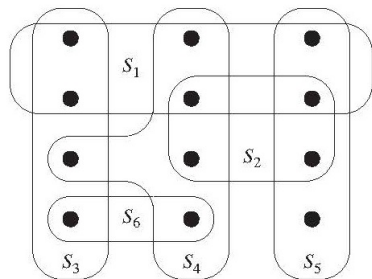
- Se trata de elegir el  $S_i$  con mayor número de elementos de entre los que quedan por cubrir

APROXSC( $S_1, \dots, S_m$ )

- 1  $U = \{S_1, \dots, S_m\}$
- 2  $W = \emptyset$
- 3  $B = S_1 \cup \dots \cup S_m$
- 4 **while**  $B \neq \emptyset$
- 5     sea  $A$  el conjunto de  $U$  con más elementos de  $B$
- 6     Borrar  $A$  de  $U$
- 7     Añadir  $A$  a  $W$
- 8     Borrar todos los elementos en  $A$  de  $B$
- 9 Resultado  $W$

- Tiempo polinómico en  $n$  y  $m$  (mín( $n, m$ ) iteraciones de  $O(nm)$ , con  $n = |S_1 \cup \dots \cup S_m|$ )

# Ejemplo de aplicación del algoritmo



$ U $	12	6	3	1	0
$ S_1 $	<b>6</b>	X	X	X	X
$ S_2 $	4	2	1	X	X
$ S_3 $	4	2	1	<b>1</b>	X
$ S_4 $	5	<b>3</b>	X	X	X
$ S_5 $	4	2	<b>2</b>	X	X
$ S_6 $	2	2	1	1	X

## Ejemplo de aplicación del algoritmo

El resultado del algoritmo es  $k = 4$ ,  $W = \{S_1, S_4, S_5, S_3\}$

# Algoritmo de aproximación de Set cover (voraz)

- Se trata de elegir el  $S_i$  con mayor número de elementos de entre los que quedan por cubrir

APROXSC( $S_1, \dots, S_m$ )

- 1  $U = \{S_1, \dots, S_m\}$
- 2  $W = \emptyset$
- 3  $B = S_1 \cup \dots \cup S_m$
- 4 **while**  $B \neq \emptyset$
- 5     sea  $A$  el conjunto de  $U$  con más elementos de  $B$
- 6     Borrar  $A$  de  $U$
- 7     Añadir  $A$  a  $W$
- 8     Borrar todos los elementos en  $A$  de  $B$
- 9 Resultado  $W$



# Análisis del algoritmo (delicado)

- Es una  $\log n$ -aproximación, con  $n = |S_1 \cup \dots \cup S_m|$ :
  - ▶ Los  $A$  elegidos cubren cada vez menos elementos de  $B$
  - ▶ Vamos a fijarnos en cuando quedan en  $B$ :  $2^{\lceil \log n \rceil - 1}, \dots, 2^i, \dots$
  - ▶ Sea  $w_i$  el número de  $A$ s seleccionados desde la primera vez que quedan  $\leq 2^{i+1} - 1$  hasta la primera vez que quedan  $\leq 2^i$  (Fase  $i$ )
  - ▶  $w = \max w_i$
  - ▶  $\hat{c}(S_1, \dots, S_m) \leq w \log n$

- **También se cumple** que  $c(S_1, \dots, S_m) \geq w$ 
  - ▶ En la fase  $i$  se cubren unos  $2^i$  elementos (desde  $\leq 2^{i+1} - 1$  hasta  $\leq 2^i$ )
  - ▶ Los conjuntos elegidos cubren cada vez menos elementos
  - ▶ Nos quedamos justo antes de coger el último  $A$  de la fase  $i$ , llamamos  $B'$  a ese  $B$
  - ▶ El último  $A$  seleccionado en la fase  $i$  cubre  $\leq 2^i/w_i$  elementos de  $B'$  (porque es el que menos cubre y entre  $w_i$  conjuntos cubren  $2^i$  elementos)
  - ▶ No existe ningún  $S_r$  que cubra más de  $2^i/w_i$  elementos de  $B'$
  - ▶ Luego necesito al menos  $w_i$  conjuntos  $S_r$  que cubran  $B'$
  - ▶ Luego  $c(S_1, \dots, S_m) \geq w_i$
  - ▶ Como se cumple para cualquier  $i$ ,  $c(S_1, \dots, S_m) \geq w$  ( $w = \max w_i$ )

# Análisis del algoritmo (delicado)

- Hemos visto  $\hat{c}(S_1, \dots, S_m) \leq w \log n$
- También que  $c(S_1, \dots, S_m) \geq w$
- $\hat{c}(S_1, \dots, S_m) \leq w \log n \leq c(S_1, \dots, S_m) \log n$
- Luego  $\hat{c}(S_1, \dots, S_m)/c(S_1, \dots, S_m) \leq \log n$
- El análisis es exacto, hay casos en que  $\hat{c}(S_1, \dots, S_m)/c(S_1, \dots, S_m) = \log n$

- Lo anterior da una aproximación muy mala (???) (ratio no constante)  
...
- No se puede aproximar con ratio  $1 + \epsilon$  para ningún  $\epsilon$  (si  $P \neq NP$ )

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 **Mochila**
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 Resumen

- Recordemos **el problema de la mochila**:
  - ▶ Se tienen  $n$  objetos NO fraccionables y una mochila.
  - ▶ El objeto  $i$  tiene peso  $p_i \in \mathbb{R}^+$  y produce beneficio  $b_i \in \mathbb{R}^+$
  - ▶ El objetivo es llenar la mochila, de capacidad  $C \in \mathbb{R}^+$ , de manera que se maximice el beneficio

*Problema:* **Mochila**

*Entrada:*  $n \in \mathbb{N}$ ,  $p_1, \dots, p_n \in \mathbb{R}^+$ ,  $b_1, \dots, b_n \in \mathbb{R}^+$ ,  $C \in \mathbb{R}^+$ .

*Salida:* Encontrar  $x_1, \dots, x_n \in \{0, 1\}$  (indicando que el objeto se deja fuera o se mete en la mochila) con

$$\sum_{1 \leq i \leq n} b_i x_i$$

el máximo posible tal que

$$\sum_{1 \leq i \leq n} p_i x_i \leq C$$

## Solución con programación dinámica (AB)

- Restringida a pesos  $p_i$  y capacidad  $C$  números naturales
- Consiste en considerar la  $n \times (C + 1)$  matriz  $g$  con valores

$g[i, x] =$  máximo beneficio con los objetos del 1 al  $i$  y con capacidad  $x$

- (El beneficio de) la solución buscada es  $g[n, C]$
- Para “rellenar” la matriz basta con rellenar la primera fila y la primera columna y después aplicar la regla:

$$g[j, x] = \max(g[j - 1, x], g[j - 1, x - p_j] + b_j)$$

- A partir de la matriz es fácil reconstruir los objetos para obtener  $g[n, C]$
- Tiempo  $O(nC)$ , puede ser enorme dependiendo de  $C$

# Solución con programación dinámica

DINAM( $n, p_1, \dots, p_n, b_1, \dots, b_n, C$ )

```
1  for  $x=1$  to  $p_1-1$ 
2       $g[1, x] = 0$ 
3  for  $x=p_1$  to  $C$ 
4       $g[1, x] = b_1$ 
5  for  $j=1$  to  $n$ 
6       $g[j, 0] = 0$ 
7  for  $j=2$  to  $n$ 
8      for  $x=1$  to  $C$ 
9           $g[j, x] = \text{máx}(g[j-1, x], g[j-1, x-p_j] + b_j)$ 
10 Resultado  $g[n, C]$ 
```



## Otra solución con programación dinámica: **Dinam2**

- Restringida a beneficios  $b_i$ ; números naturales
- Sea  $B$  el máximo de todos los beneficios,  $M = nB$
- Consiste en considerar la  $n \times (M + 1)$  matriz  $U$  con valores

$U[i, x] =$  mínimo peso con los objetos del 1 al  $i$  y con beneficio  $x$

- La solución buscada tiene beneficio  $x$  con  $x$  el mayor tal que  $U[n, x] \leq C$
- Para “rellenar” la matriz basta con rellenar la primera fila y la primera columna y después aplicar la regla:

$$U[j, x] = \min(U[j - 1, x], U[j - 1, x - b_j] + p_j)$$

- A partir de la matriz es fácil reconstruir los objetos para obtener la solución
- Tiempo  $O(nM)$ , puede ser enorme dependiendo de  $M$

## Otra solución con programación dinámica

DINAM2( $n, p_1, \dots, p_n, b_1, \dots, b_n, C$ )

```
1   $B = \max(b_1, \dots, b_n)$ 
2   $M = nB$ 
3  for  $x=1$  to  $b_1$ 
4       $U[1, x] = p_1$ 
5  for  $x=b_1+1$  to  $M$ 
6       $U[1, x] = \infty$ 
7  for  $j=1$  to  $n$ 
8       $U[j, 0] = 0$ 
9  for  $j=2$  to  $n$ 
10     for  $x=1$  to  $M$ 
11          $U[j, x] = \min(U[j-1, x], U[j-1, x-b_j] + p_j)$ 
12  Resultado  $\max\{x \mid U[n, x] \leq C\}$ 
```

# Solución redondeando

- Vamos a utilizar la segunda solución con programación dinámica pero para un problema simplificado
- Fijamos un  $R > 0$ , **vamos a redondear los beneficios** dividiendo por  $R$
- Sea  $n \in \mathbb{N}$ ,  $p_1, \dots, p_n \in \mathbb{R}^+$ ,  $C \in \mathbb{R}^+$ ,  $b_1, \dots, b_n \in \mathbb{R}^+$ , como en la entrada original y  $\hat{b}_1, \dots, \hat{b}_n \in \mathbb{N}$  los beneficios (naturales) definidos como sigue:

$$\hat{b}_i = \lfloor b_i/R \rfloor$$

- Es decir, cambiamos de escala los beneficios
- Como  $\max \hat{b}_i \leq B/R$  ( $B$  es el máximo de los  $b_i$ ), en tiempo  $O(n^2 B/R)$  resolvemos el problema simplificado

# Solución redondeando

REDONDMOCHILA( $n, p_1, \dots, p_n, b_1, \dots, b_n, C, R$ )

1  $B = \text{máx}(b_1, \dots, b_n)$

2  $\hat{b}_i = \lfloor b_i/R \rfloor$

3  $(x_1, \dots, x_n) = \text{Dinam2}(n, p_1, \dots, p_n, \hat{b}_1, \dots, \hat{b}_n, C)$

4 Resultado  $x_1, \dots, x_n$

# Solución redondeando

- $\hat{b}_i = \lfloor b_i/R \rfloor$
- En tiempo  $O(n^2B/R)$  resolvemos el problema simplificado
- Pero un beneficio  $\hat{x}$  al problema simplificado supone un beneficio de al menos  $R\hat{x}$  al problema original

$$\hat{x} = \sum_A \hat{b}_i = \sum_A \lfloor b_i/R \rfloor$$

$$R\hat{x} = \sum_A R\lfloor b_i/R \rfloor \leq \sum_A b_i$$

- Y un beneficio  $x$  al problema original supone un beneficio de al menos  $x/R - n$  al problema simplificado

$$x = \sum_A b_i$$

$$\sum_A \hat{b}_i = \sum_A \lfloor b_i/R \rfloor \geq \sum_A (b_i/R - 1) \geq \sum_A b_i/R - n$$

# Solución redondeando

- $\hat{b}_i = \lfloor b_i/R \rfloor$
- Tiempo  $O(n^2 B/R)$
- Un beneficio  $\hat{x}$  al problema simplificado supone un beneficio de al menos  $R\hat{x}$  al problema original
- Y un beneficio  $x$  al problema original supone un beneficio de al menos  $x/R - n$  al problema simplificado

## Solución $\epsilon$ -aproximada para cualquier $\epsilon < 1$

- Fijamos  $\epsilon > 0$
- Vamos a utilizar la solución redondeada con  $R = \epsilon B / (2n)$
- Tiempo  $O(n^2 B / R) = O(n^2 B / (\epsilon B / (2n))) = O(n^3 / \epsilon)$
- Un beneficio  $\hat{x}$  al problema simplificado supone un beneficio de al menos  $R\hat{x} = \epsilon B / (2n)\hat{x}$  al problema original
- Y un beneficio  $x$  al problema original supone un beneficio de al menos  $x/R - n = x2n / (\epsilon B) - n$  al problema simplificado
- Sea  $n \in \mathbb{N}$ ,  $p_1, \dots, p_n \in \mathbb{R}^+$ ,  $C \in \mathbb{R}^+$  como en la entrada original y  $\hat{b}_1, \dots, \hat{b}_n \in \mathbb{N}$  los beneficios (naturales) definidos como sigue:

$$\hat{b}_i = \lfloor b_i / R \rfloor = \lfloor b_i 2n / (\epsilon B) \rfloor$$

# Solución $\epsilon$ -aproximada para cualquier $\epsilon < 1$

APROXMOCHILA( $n, p_1, \dots, p_n, b_1, \dots, b_n, C, \epsilon$ )

1  $B = \text{máx}(b_1, \dots, b_n)$

2  $\hat{b}_i = \lfloor b_i 2n / (\epsilon B) \rfloor$

3  $(x_1, \dots, x_n) = \text{Dinam2}(n, p_1, \dots, p_n, \hat{b}_1, \dots, \hat{b}_n, C)$

4 Resultado  $x_1, \dots, x_n$



# Análisis del algoritmo

- 1 Recordad que
  - ▶ Un beneficio  $\hat{x}$  al problema simplificado supone un beneficio de al menos  $R\hat{x} = \hat{x}\epsilon B/(2n)$  al problema original
  - ▶ Y un beneficio  $x$  al problema original supone un beneficio de al menos  $x/R - n = x2n/(\epsilon B) - n$  al problema simplificado
- 2 Si  $\hat{S}$  es el conjunto óptimo para  $(n, p_1, \dots, p_n, \hat{b}_1, \dots, \hat{b}_n, C)$ ,

$$\sum_{i \in \hat{S}} b_i \geq \epsilon B / (2n) \sum_{i \in \hat{S}} \hat{b}_i$$

- 3 Si  $S$  es el conjunto óptimo para  $(n, p_1, \dots, p_n, b_1, \dots, b_n, C)$ ,

$$\sum_{i \in S} \hat{b}_i \geq \left( \sum_{i \in S} b_i \right) 2n / (\epsilon B) - n$$

- 4 Como  $\hat{S}$  es óptimo para  $(n, p_1, \dots, p_n, \hat{b}_1, \dots, \hat{b}_n, C)$ ,

$$\sum_{i \in \hat{S}} \hat{b}_i \geq \sum_{i \in S} \hat{b}_i$$

- 4 Uniendo las tres anteriores,

$$\sum_{i \in \hat{S}} b_i \geq \epsilon B / (2n) \sum_{i \in \hat{S}} \hat{b}_i$$

$$\sum_{i \in \hat{S}} \hat{b}_i \geq \sum_{i \in S} \hat{b}_i$$

$$\sum_{i \in S} \hat{b}_i \geq \left( \sum_{i \in S} b_i \right) 2n / (\epsilon B) - n$$

- 5 tenemos

$$\begin{aligned} \frac{\sum_{i \in S} b_i}{\sum_{i \in \hat{S}} b_i} &\leq \frac{\sum_{i \in S} b_i}{\sum_{i \in S} b_i - \epsilon B / 2} = \\ &= 1 + \frac{\epsilon B / 2}{\sum_{i \in S} b_i - \epsilon B / 2} \leq 1 + \frac{\epsilon / 2}{1 - \epsilon / 2} \leq 1 + \epsilon \end{aligned}$$

Porque  $\sum_{i \in S} b_i \geq B$  y  $\epsilon < 1$

- Como hemos dicho antes, en tiempo  $O(n^3/\epsilon)$  resolvemos el problema simplificado, luego polinómico en  $n$  y en  $1/\epsilon$  **Es lo que llamamos un FPTAS**
- En general podemos utilizar la programación dinámica como aproximación:
  - 1 Reescalamos los valores para que la programación dinámica sea más rápida
  - 2 Comparamos la solución obtenida con la solución óptima

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 **Cobertura de Vértices con Pesos**
- 9 Otra vez TSP
- 10 Resumen

- Vamos a ver un algoritmo de aproximación basado en **programación lineal**
- Se trata de una 2-aproximación eficiente

# Cobertura de Vértices con Pesos

*Problema:* **wVC**

*Entrada:* Un grafo  $G$  (con vértices  $V$ ), un peso positivo  $w(v)$  para cada vértice  $v \in V$ .

*Salida:* Encontrar un conjunto  $U$  de vértices de  $G$  con el menor peso posible

$$\sum_{v \in U} w(v)$$

y tal que cada arista  $(i, j)$  de  $G$  cumple que  $i \in U$  ó  $j \in U$

- Es una generalización de VC (minimizar el número de vértices) ya que VC es el caso  $w(v) = 1$  para todo  $v$
- Al tratarse de una generalización de VC no se puede aproximar con ratio 1,1666 (si  $P \neq NP$ )

# Programa lineal para wVC

- No podemos aplicar el algoritmo aproximado para VC (no aproxima bien)
- Podemos representar el problema de optimización como un problema de programación lineal:
  - ▶ Asociamos a cada vértice  $v$  una variable  $x(v)$  que puede valer 0 ó 1
  - ▶ Añadimos  $v$  al cubrimiento si y sólo si  $x(v) = 1$
  - ▶ Para cada arista  $(u, v)$  la condición de que  $u$  ó  $v$  deben estar en el cubrimiento se escribe

$$x(u) + x(v) \geq 1$$

- ▶ Así tenemos el siguiente **programa entero 0-1** para encontrar el mínimo cubrimiento:

<b>Función Objetivo</b>	$\text{mín } \sum_{v \in V} w(v)x(v)$	
<b>Restricciones</b>	$x(u) + x(v) \geq 1$	para cada arista $(u, v)$
	$x(v) \in \{0, 1\}$	para cada vértice $v$

- Si quitamos la restricción  $x(v) \in \{0, 1\}$  y la sustituimos por  $0 \leq x(v) \leq 1$  obtenemos el siguiente **programa lineal (la relajación)**:

**Función Objetivo**    mín  $\sum_{v \in V} w(v)x(v)$

**Restricciones**     $x(u) + x(v) \geq 1$             para cada arista  $(u, v)$   
                          $0 \leq x(v) \leq 1$             para cada vértice  $v$

- El problema original es un caso particular de la relajación, luego el óptimo de la relajación es una cota inferior del óptimo del original
- ¿Cómo podemos usar la relajación para aproximar el original?



# Algoritmo aproximado para wVC

APROX-MIN-WEIGHT-VC( $G, w$ )

```
1  $U = \emptyset$ 
2 Calcula  $\hat{x}$ , una solución óptima para el programa lineal relajación
3 for  $v \in V$ 
4     if  $\hat{x}(v) \geq 1/2$ 
5     Añadir  $v$  a  $U$ 
6 Resultado  $U$ 
```

- La solución obtenida es un “redondeo” de la solución del programa lineal relajado
- ¿Cómo de buena es la aproximación?

## 2-aproximación para wVC

- Sea  $c(G, w)$  una solución óptima para wVC
- Sea  $\hat{x}$  una solución óptima para el programa lineal relajación
- Sea  $z = \sum_{v \in V} w(v)\hat{x}(v)$
- Como un cubrimiento es una solución del programa lineal relajación,

$$z \leq c(G, w)$$

## 2-aproximación para wVC

- Redondeando para encontrar el conjunto  $U$  (cogiendo los  $v$  con  $\hat{x}(v) \geq 1/2$ ) veamos que obtenemos un cubrimiento válido con peso  $w(U) \leq 2z$ :
  - ▶  $U$  es un cubrimiento válido porque para cada arista  $(u, v)$ ,  $\hat{x}(u) + \hat{x}(v) \geq 1$ , luego al menos uno de entre  $\hat{x}(u)$  y  $\hat{x}(v)$  es al menos  $1/2$  y es incluido en  $U$

## 2-aproximación para wVC

- Tenemos

$$\begin{aligned}z &= \sum_{v \in V} w(v) \hat{x}(v) \\ &\geq \sum_{v \in V, \hat{x}(v) \geq 1/2} w(v) \hat{x}(v) \\ &\geq \sum_{v \in V, \hat{x}(v) \geq 1/2} w(v) 1/2 \\ &= \sum_{v \in U} w(v) 1/2 \\ &= 1/2 \cdot w(U) = 1/2 \cdot \hat{c}(G, w)\end{aligned}$$

- Luego  $\hat{c}(G, w) \leq 2z \leq 2c(G, w)$
- Y Aprox-Min-Weight-VC es una 2-aproximación para wVC

# Complejidad de la aproximación

- Existen algoritmos en tiempo polinómico para programación lineal (algoritmo de Karmarkar)
- También visteis el Simplex que es exponencial pero en muchos casos es rápido
- Luego podemos 2-aproximar wVC **en tiempo polinómico**

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 **Otra vez TSP**
- 10 Resumen

*Problema:* TSP minimización

*Entrada:*  $n$  el número de ciudades y  $M$  la matriz de distancias  $n \times n$ .

*Salida:* Encontrar un recorrido por las  $n$  ciudades, sin repeticiones y volviendo al punto de partida con distancia total la mínima posible

- El mayor problema resuelto óptimamente hasta 2006 es una entrada de 85.900 ciudades
- Usando heurísticas, varias entradas de millones de ciudades se han resuelto dentro del 1 % de la solución óptima
- Las distancias no tienen que satisfacer la desigualdad triangular para todas las ciudades
  - ▶ Piensa en la distancia definida como tiempo de vuelo entre las dos ciudades
  - ▶ O simplemente en el camino que selecciona un GPS con la opción “Ruta más rápida”
  - ▶ O el precio de volar de una ciudad a otra ...

# TSP no se puede aproximar

- Hemos visto problemas para los que conseguíamos una aproximación de ratio constante
- Otros para los que conseguíamos aproximación todo lo pequeña que queramos
- Para TSP el resultado es negativo, no se puede aproximar para ninguna constante **(Si  $P \neq NP$ )**



# TSP no se puede aproximar

- Recordemos que *Ciclo Hamiltoniano* es NP-difícil  
*Problema:* **Ciclo Hamiltoniano**  
*Entrada:* Un grafo  $G$  con  $n$  vértices.  
*Salida:* ¿Existe un ciclo hamiltoniano en  $G$ , es decir, un camino que visita una vez cada vértice y vuelve al vértice inicial?
- Vamos a demostrar que **si podemos  $C$ -aproximar *TSP* minimización para alguna constante  $C$  entonces podemos resolver *Ciclo Hamiltoniano* eficientemente** (en tiempo polinómico)
- Luego **si  $P \neq NP$  entonces *Ciclo Hamiltoniano* no se puede resolver eficientemente y por tanto no podemos aproximar *TSP* minimización con ningún ratio constante**

# si podemos $C$ -aproximar TSP entonces podemos resolver HAM

- Supongamos que tenemos un algoritmo eficiente  $AproxTSP$  que  $C$ -aproxima  $TSP$  *minimización*
- Dada una entrada  $G$  de *Ciclo Hamiltoniano* la convertimos en una entrada de TSP, con distancia 1 cuando había arista en  $G$  y distancia  $Cn + 1$  cuando no había
- Veremos que a partir de un recorrido  $C$ -aproximado podemos saber si existe un ciclo hamiltoniano para el grafo  $G$  original

# si podemos $C$ -aproximar TSP entonces podemos resolver HAM

CICLOHAMILTONIANO( $G$ )

```
1   $n =$  número de vértices de  $G$ 
2  for  $i = 1$  to  $n$ 
3      for  $j = 1$  to  $n$ 
4          if  $(i, j)$  es arista de  $G$ 
5               $d(i, j) = 1$ 
6          else  $d(i, j) = Cn + 1$ 
7       $R = \text{AproxTSP}(n, d)$ .
8      if  $\sum_{i=1}^{n-1} d(R[i], R[i + 1]) \leq Cn$ 
9          Resultado Cierto
10     else Resultado Falso
```

- Si  $G$  tiene hamiltoniano entonces el recorrido óptimo de  $(n, d)$  medirá  $n$  y  $AproxTSP$  devuelve un recorrido de distancia total  $\leq Cn$  y el algoritmo devuelve Cierto
- Si  $G$  no tiene ciclo hamiltoniano entonces cualquier recorrido sin repeticiones de todas las ciudades de  $(n, d)$  medirá al menos  $(n - 1) + (Cn + 1) = (C + 1)n$  y el algoritmo devuelve Falso

- Hemos reducido *Ciclo Hamiltoniano* a *C*-aproximar TSP
- Luego hemos visto que para cada *C*, ***C*-aproximar TSP es NP-difícil**
- Es lo mismo que pasaba para los problemas:
  - ▶ 1,166-aproximar VC es NP-difícil
  - ▶ 1,013-aproximar *TSP métrico* es NP-difícil
  - ▶ para cada *C*, *C*-aproximar *Set Cover* es NP-difícil

# Contenido de este tema

- 1 Introducción
- 2 Problemas de optimización
- 3 Diferentes tipos de aproximación
- 4 Cobertura de Vértices
- 5 TSP métrico
- 6 Set cover
- 7 Mochila
- 8 Cobertura de Vértices con Pesos
- 9 Otra vez TSP
- 10 **Resumen**

- Hemos visto algoritmos de aproximación con ratio
  - ▶ logarítmico
  - ▶ constante
  - ▶ cualquier constante
- Estos son los problemas concretos:
  - ▶ log-aproximación de *Set Cover*
  - ▶ 2-aproximación de VC
  - ▶ 2-aproximación de wVC
  - ▶ 2-aproximación de *TSP métrico* (también existe 1,5-aproximación)
  - ▶  $C$ -aproximación de *Mochila* para cualquier  $C$
- Resultados negativos: si  $P \neq NP$  entonces ...
  - ▶ No se puede 1,013-aproximar *TSP métrico*
  - ▶ No se puede 1,1666-aproximar VC
  - ▶ No se puede 1,1666-aproximar wVC
  - ▶ No se puede  $C$ -aproximar *Set Cover* para ninguna  $C$
  - ▶ No se puede  $C$ -aproximar *TSP* para ninguna  $C$

- Hemos usado como técnicas
  - ▶ algoritmos voraces
  - ▶ programación dinámica
  - ▶ programación lineal
- Estos son los problemas concretos:
  - ▶ VC aproximación voraz
  - ▶ *Set Cover* aproximación voraz
  - ▶ *Mochila* programación dinámica
  - ▶ wVC programación lineal
  - ▶ TSP métrico spanning trees y recorridos eulerianos (perfect matching)