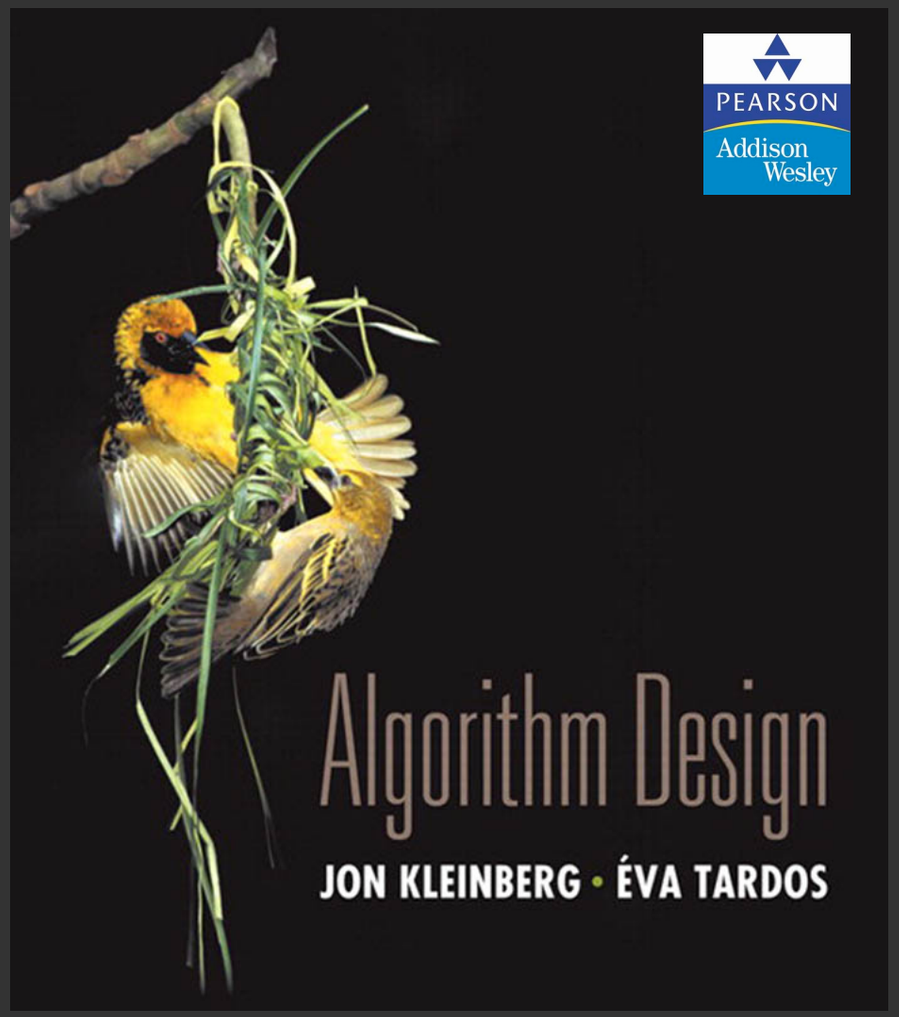


8. Intrattabilità III

- *casi particolari: alberi*
- *casi particolari: planarità*
- *algoritmi approssimanti: vertex cover*
- *algoritmi approssimanti: bisaccia*
- ~~*algoritmi esponenziali: 3-SAT*~~
- ~~*algoritmi esponenziali: TSP*~~

Traduzione e adattamento di Vincenzo Bonifaci
Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on 30/05/2022 11:51

Come affrontare i problemi NP-ardui

D. Devo risolvere un problema **NP**-arduo. Cosa dovrei fare?

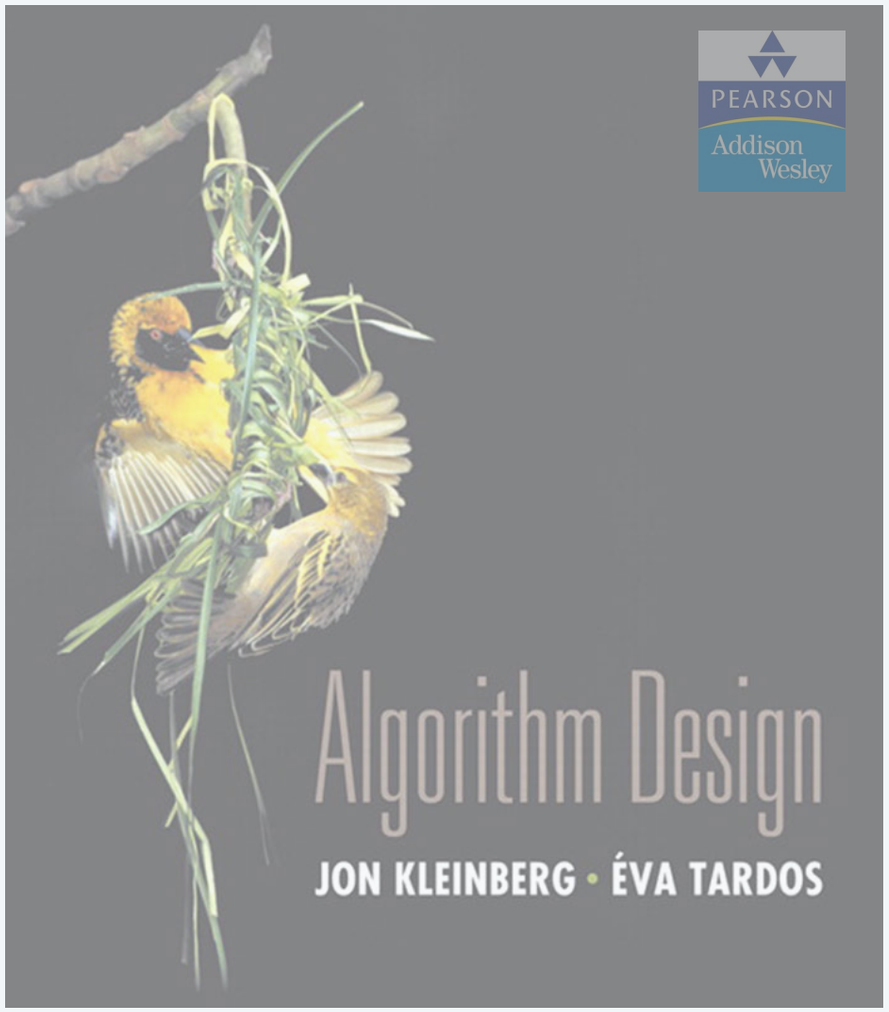
R. Dovrai sacrificare una delle caratteristiche desiderabili dell'algoritmo:

i. Risolve istanze arbitrarie del problema.
ii. Risolve il problema in modo ottimo.
iii. Risolve il problema in tempo polinomiale.

Strategie per affrontare problemi NP-ardui.

i. Progettare algoritmi per casi particolari del problema.
ii. Progettare algoritmi di approssimazioni o euristiche.
iii. Progettare algoritmi tempo esponenziali.

possiamo usare algoritmi avidi, programmazione dinamica, divide et impera, e flussi di rete!



SECTION 10.2

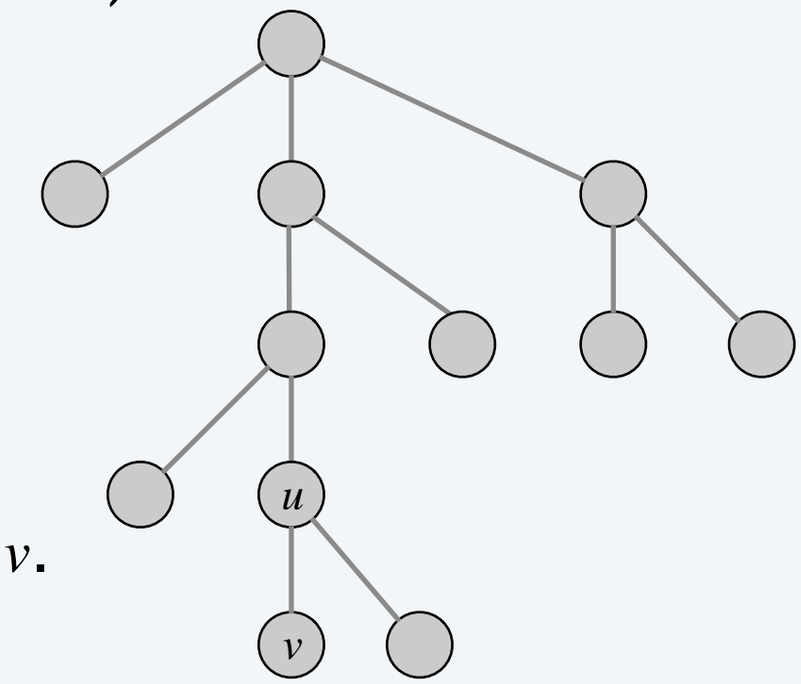
8. INTRATTABILITÀ III

- *casi particolari: alberi*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

Insieme indipendente [Independent set] su alberi

Independent set su alberi. Dato un albero, trova un sottoinsieme di nodi a massima cardinalità tale che nessuna coppia di nodi del sottoinsieme sia adiacente.

Fatto. Ogni albero ha almeno un nodo foglia (grado = 1).



Osservazione chiave. Se il nodo v è foglia, esiste un independent set a massima cardinalità che contiene v.

Dim. [argomentazione basata su scambi]

- Consideriamo un independent set S a massima cardinalità.
- Se $v \in S$, abbiamo la tesi.
- Altrimenti, sia (u, v) l'unico arco incidente a v.
  - Se $u \notin S$ e $v \notin S$, allora $S \cup \{ v \}$ è indipendente $\Rightarrow$ S non è massimo
  - Se $u \in S$ e $v \notin S$, allora $S \cup \{ v \} - \{ u \}$ è indipendente ▪

Independent set su alberi: algoritmo avido

Teorema. Il seguente algoritmo avido trova un independent set a massima cardinalità su qualunque foresta (e quindi su qualunque albero).

Dim. La correttezza segue dalla precedente osservazione. ▪

INDEPENDENT-SET-IN-A-FOREST(F)

$S \leftarrow \emptyset$.

WHILE (F ha almeno 1 arco)

  Sia v un nodo foglia e sia (u, v) l'unico arco incidente a v.

  $S \leftarrow S \cup \{ v \}$.

  $F \leftarrow F - \{ u, v \}$. ⟵ rimuovi sia u che v (e gli archi incidenti ad essi)

RETURN $S \cup \{$ nodi rimanenti in $F \}$.

Nota. Si può implementare in tempo $O(n)$ mantenendo i gradi dei nodi e una lista dei nodi di grado 1.

Intrattabilità III: quiz 1

In che modo può fallire tale algoritmo avido quando il grafo non è un albero o una foresta?

A. Può rimanere bloccato.

B. Può richiedere tempo esponenziale

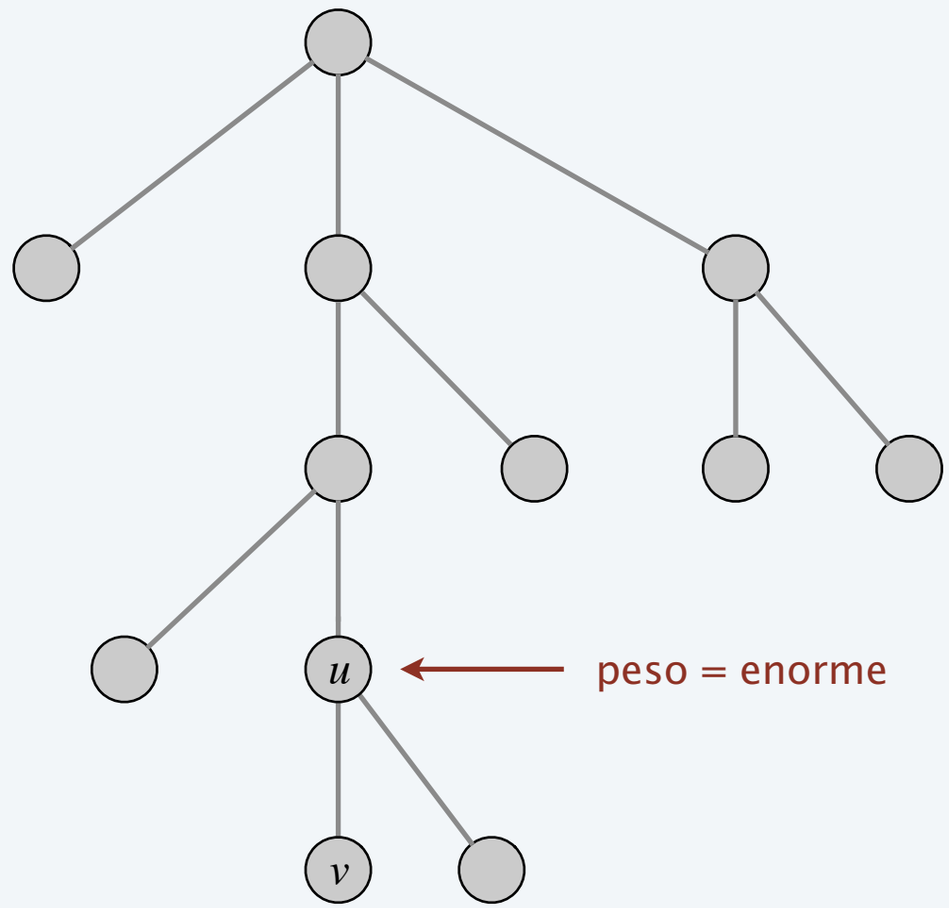
C. Può produrre un independent set non massimo.

D. Qualunque delle possibilità precedenti.

Independent set pesato su alberi

Independent set pesato su alberi. Dato un albero e pesi dei nodi $w_v \geq 0$, trova un independent set S che massimizza $\Sigma_{v \in S} w_v$.

L'algoritmo avido può fallire catastroficamente.



Independent set pesato su alberi

Independent set pesato su alberi. Dato un albero e pesi dei nodi $w_v \geq 0$, trova un independent set S che massimizza $\Sigma_{v \in S} w_v$.

Algoritmo di programmazione dinamica. Radica l'albero in un nodo r.

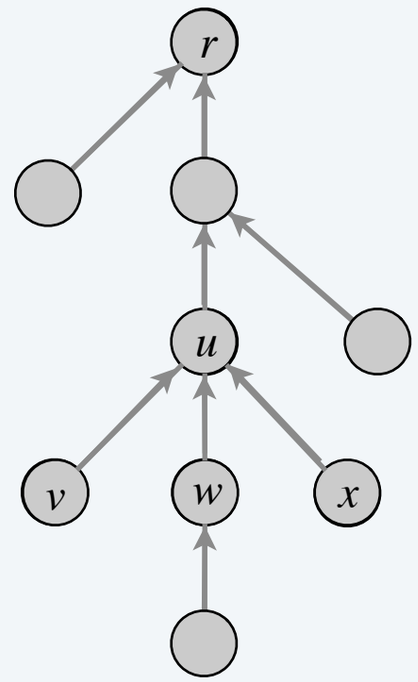
- $OPT_{in}(u)$ = IS a peso massimo nel sottoalbero radicato in u, contenente u.
- $OPT_{out}(u)$ = IS a peso massimo nel sottoalbero radicato in u, non contenente u.
- Goal: $\max \{ OPT_{in}(r), OPT_{out}(r) \}$.

Equazione di Bellman.

sottoproblemi sovrapposti

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max \{ OPT_{in}(v), OPT_{out}(v) \}$$



children(u) = { v, w, x }

Intrattabilità III: quiz 2

In quale ordine vanno risolti i sottoproblemi?

A. In preordine.

B. In postordine.

C. In ordine di livello.

D. Qualunque delle precedenti.

Independent set pesato su alberi: algoritmo di programm. dinamica

Teorema. Il seguente algoritmo calcola il peso massimo di un independent set di un albero in tempo $O(n)$.

possiamo trovare anche l'independent set stesso (non solo il suo valore) con semplici aggiunte

WEIGHTED-INDEPENDENT-SET-IN-A-TREE (T)

Radica l'albero T in un qualunque nodo r.

$S \leftarrow \emptyset$.

FOREACH (nodo u di T in postordine/ordine topologico)

assicura che ogni nodo sia processato dopo tutti i suoi discendenti

  IF (u è un nodo foglia)

    $M_{in}[u] = w_u$.

    $M_{out}[u] = 0$.

  ELSE

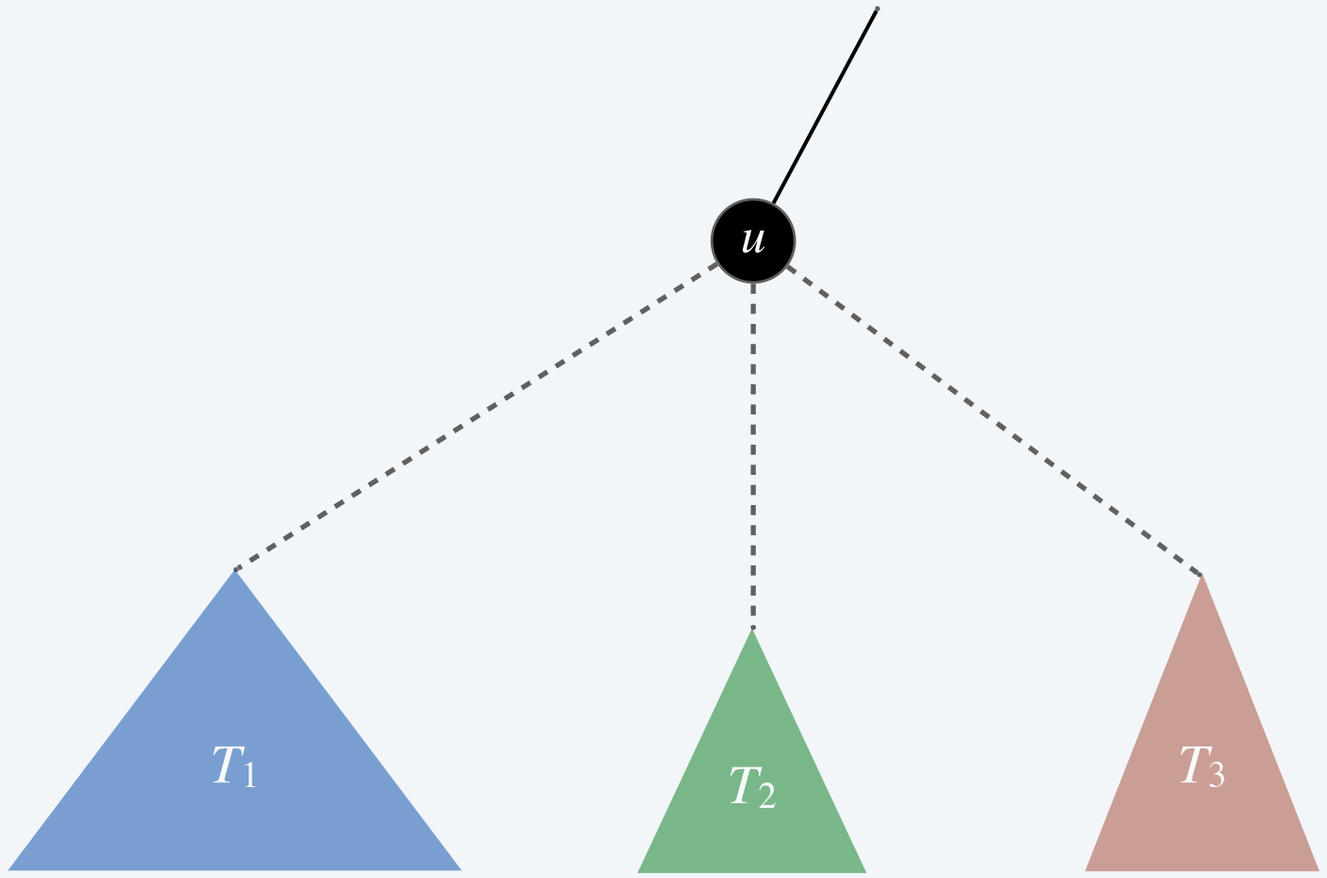
    $M_{in}[u] = w_u + \sum_{v \in children(u)} M_{out}[v]$.

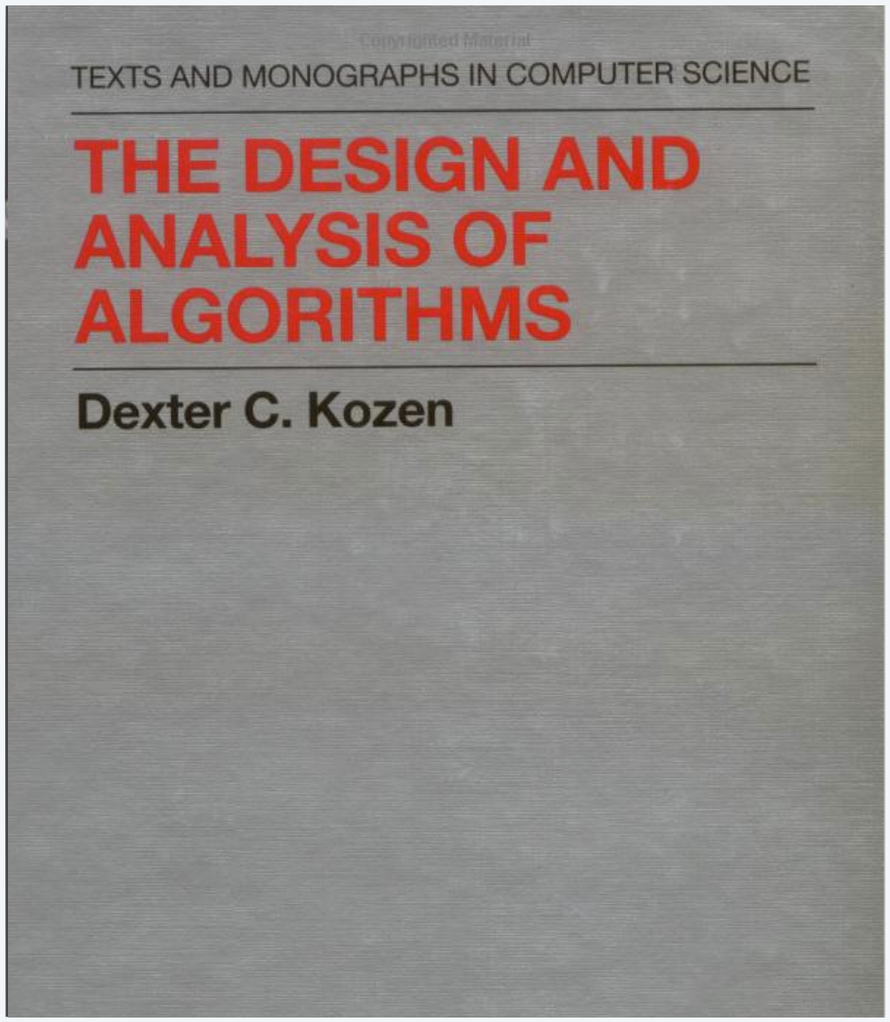
    $M_{out}[u] = \sum_{v \in children(u)} max \{ M_{in}[v], M_{out}[v] \}$.

RETURN $max \{ M_{in}[r], M_{out}[r] \}$.

Problemi NP-ardui su alberi: contesto

Independent set su alberi. Trattabile perché si può trovare un nodo che "spezza" l'interazione tra i diversi sottoproblemi legati ai diversi sottoalberi.





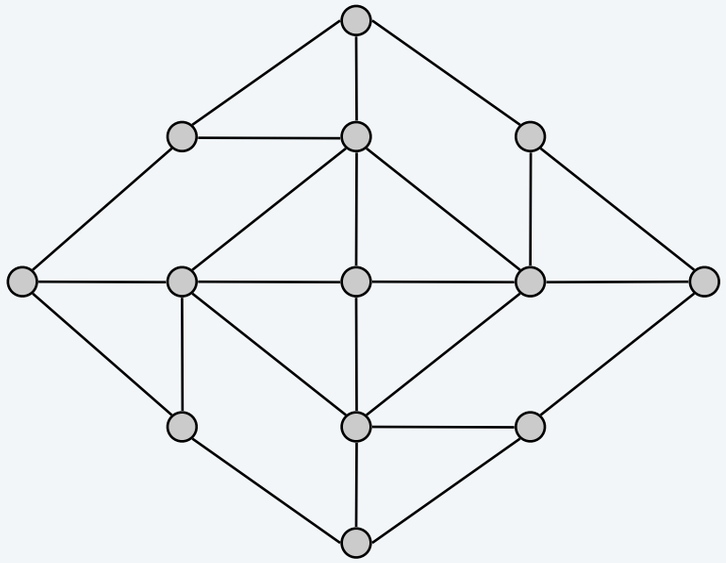
SECTION 23.1

INTRATTABILITÀ III

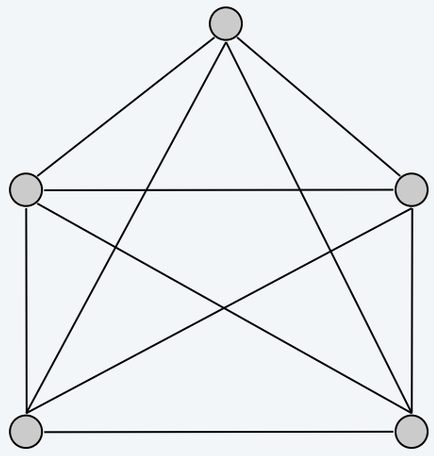
- *special cases: trees*
- *casi particolari: planarità*
- *approximation algorithms: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

Planarità

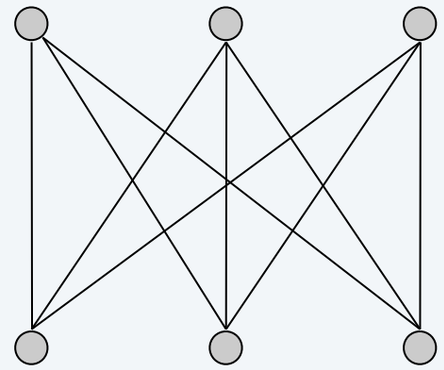
Def. Un grafo è planare se può essere immerso nel piano in modo tale che i suoi archi non si incrocino.



planare



K_5 è nonplanare



$K_{3,3}$ è nonplanare

Applicazioni. Progetto di circuiti integrati, computer grafica, ...

Test di planarità

Teorema. [Hopcroft–Tarjan 1974] Esiste un algoritmo $O(n)$ per determinare se un grafo è planare.

un grafo planare semplice ha $\leq 3n - 6$ archi

Efficient Planarity Testing

JOHN HOPCROFT AND ROBERT TARJAN

Cornell University, Ithaca, New York

ABSTRACT. This paper describes an efficient algorithm to determine whether an arbitrary graph G can be embedded in the plane. The algorithm may be viewed as an iterative version of a method originally proposed by Auslander and Parter and correctly formulated by Goldstein. The algorithm uses depth-first search and has $O(V)$ time and space bounds, where V is the number of vertices in G. An ALGOL implementation of the algorithm successfully tested graphs with as many as 900 vertices in less than 12 seconds.

Problemi su grafi planari

Fatto 0. Molti problemi su grafi possono essere risolti più efficientemente su grafi planari.

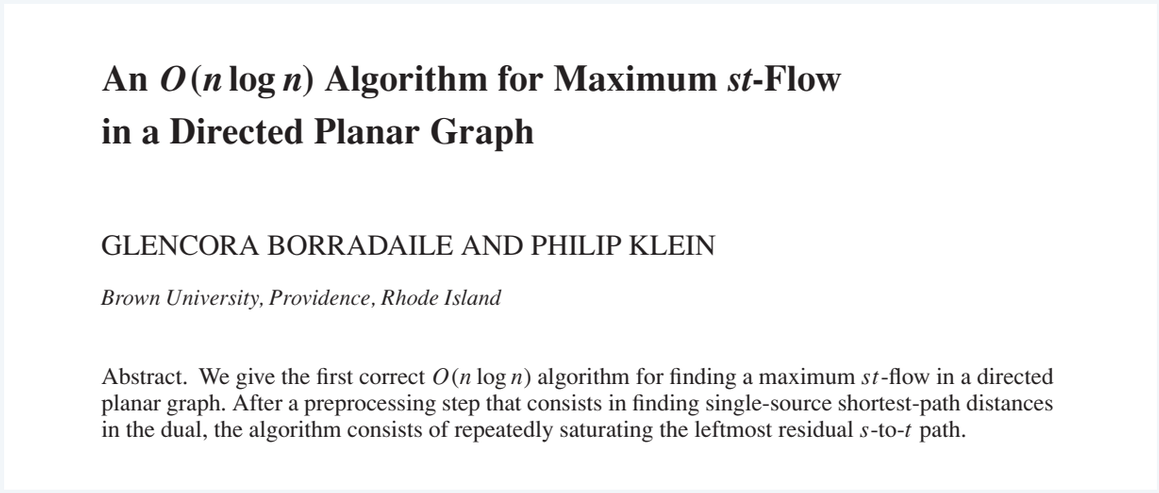
Ex. Cammini minimi, flussi massimi, MST, abbinamenti, ...

Fatto 1. Alcuni problemi **NP**-completi divengono trattabili su grafi planari.

Ex. MAX-CUT, ISING, CLIQUE, GRAPH-ISOMORPHISM, 4-COLOR, ...

Fatto 2. Altri problemi **NP**-completi si semplificano su grafi planari.

Ex. INDEPENDENT-SET, VERTEX-COVER, TSP, STEINER-TREE, ...

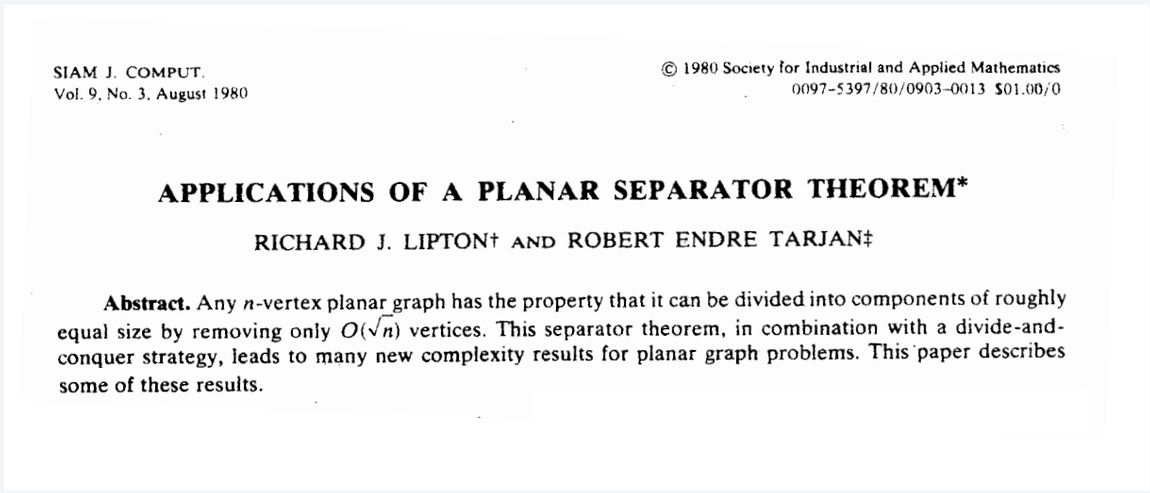


An $O(n \log n)$ Algorithm for Maximum st-Flow in a Directed Planar Graph

GLENCORA BORRADAILE AND PHILIP KLEIN

Brown University, Providence, Rhode Island

Abstract. We give the first correct $O(n \log n)$ algorithm for finding a maximum st-flow in a directed planar graph. After a preprocessing step that consists in finding single-source shortest-path distances in the dual, the algorithm consists of repeatedly saturating the leftmost residual s-to-t path.



SIAM J. COMPUT.
Vol. 9, No. 3, August 1980

0097-5397/80/0903-0013 $01.00/0

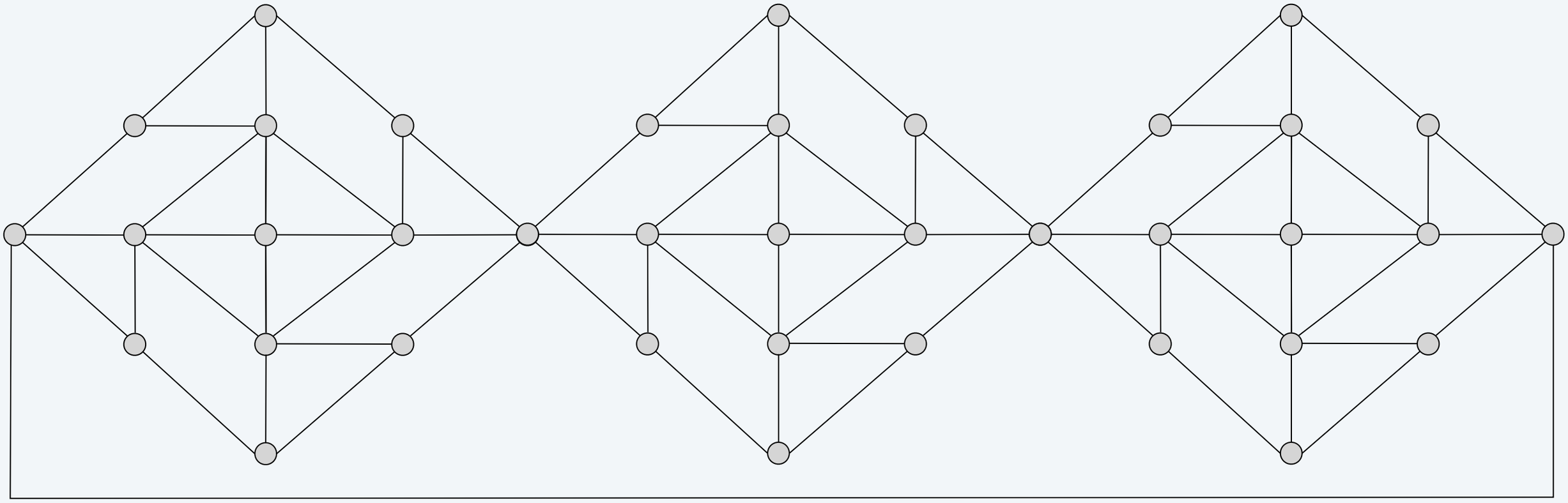
APPLICATIONS OF A PLANAR SEPARATOR THEOREM*

RICHARD J. LIPTON† AND ROBERT ENDRE TARJAN‡

Abstract. Any n-vertex planar graph has the property that it can be divided into components of roughly equal size by removing only $O(\sqrt{n})$ vertices. This separator theorem, in combination with a divide-and-conquer strategy, leads to many new complexity results for planar graph problems. This paper describes some of these results.

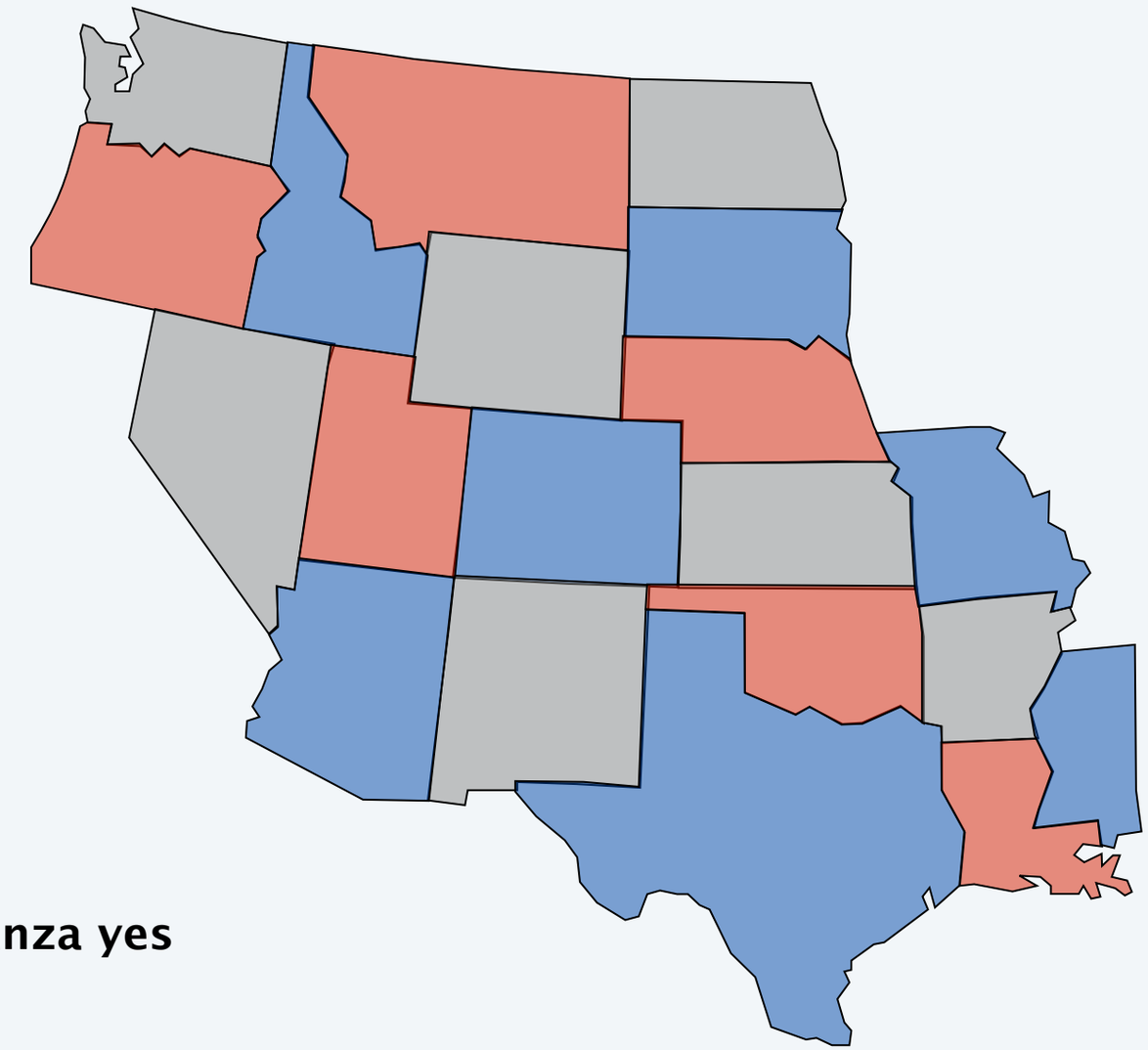
3-colorabilità di grafi planari

PLANAR-3-COLOR. Dato un grafo planare, può essere colorato con 3 colori in modo che nessuna coppia di nodi adiacenti abbia lo stesso colore?



3-colorabilità di mappe planari

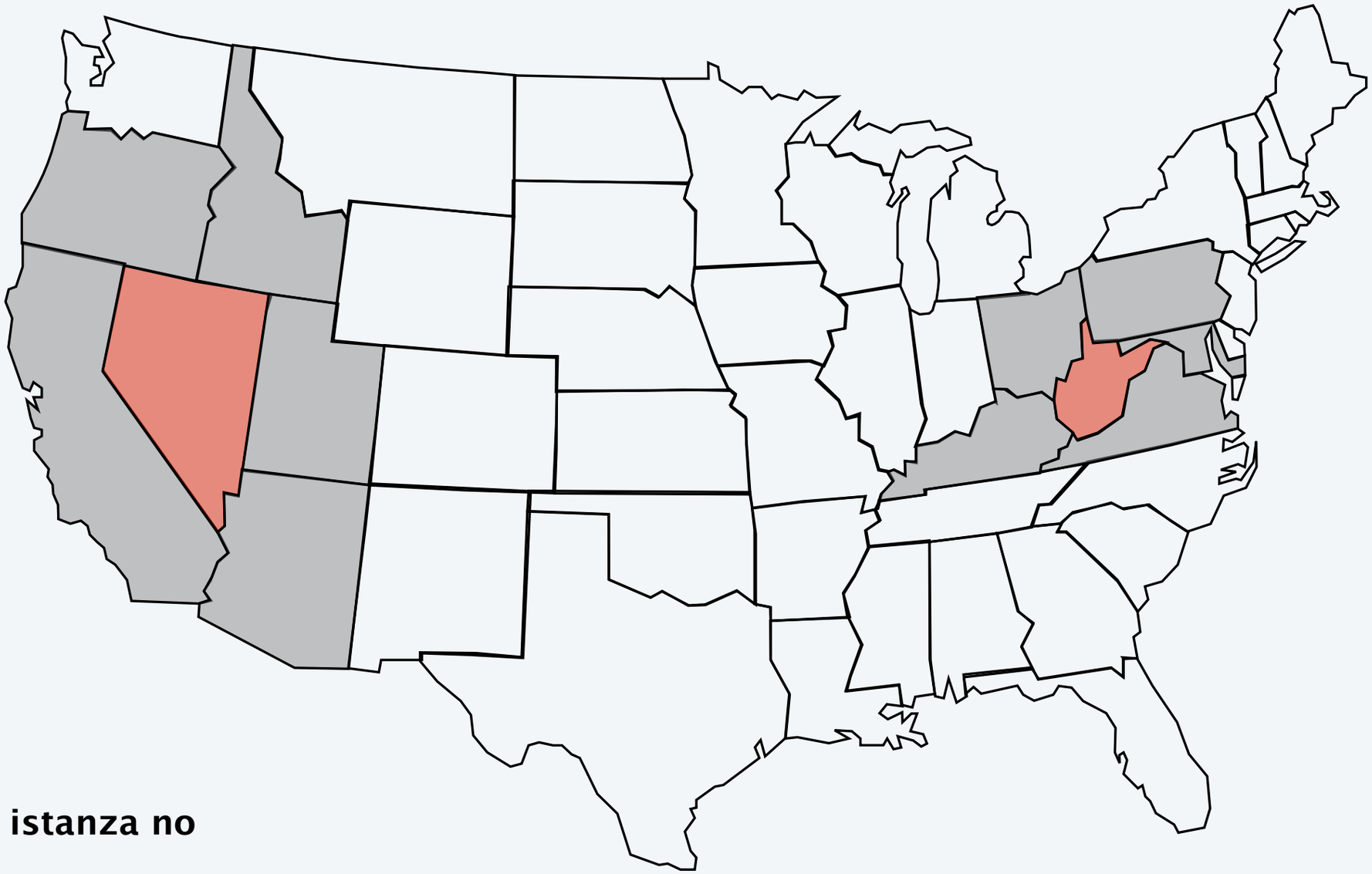
PLANAR-MAP-3-COLOR. Data una mappa planare, la si può colorare con 3 colori in modo che nessuna coppia di regioni adiacenti abbia lo stesso colore?



istanza yes

3-colorabilità di mappe planari

PLANAR-MAP-3-COLOR. Data una mappa planare, la si può colorare con 3 colori in modo che nessuna coppia di regioni adiacenti abbia lo stesso colore?



istanza no

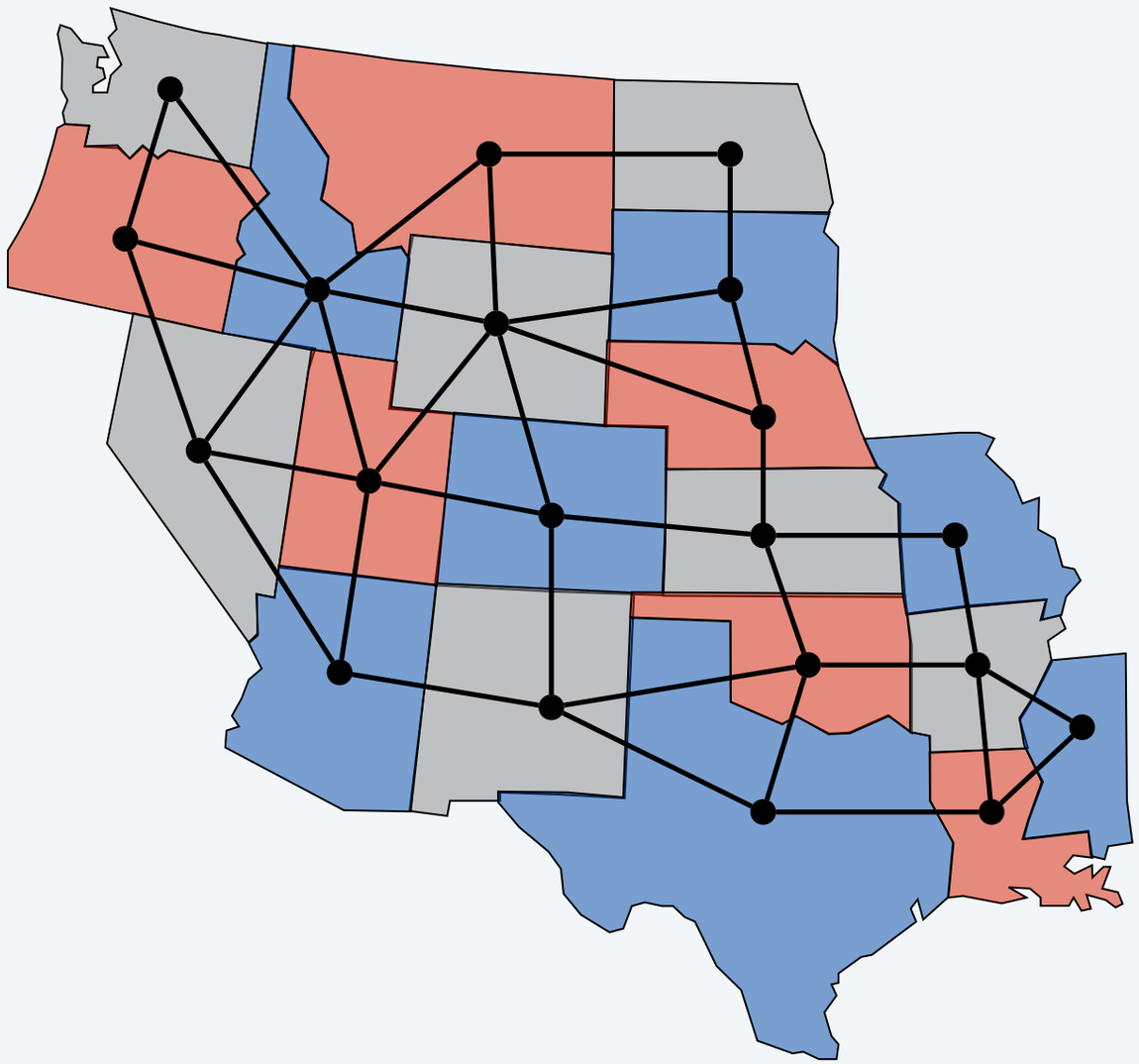
3-colorabilità di grafi planari vs. 3-colorabilità di mappe planari

Teorema. PLANAR-3-COLOR $\equiv_P$ PLANAR-MAP-3-COLOR.

Idea della dim.

- Nodi corrispondono alle regioni.
- Due nodi sono adiacenti sse essi condividono un confine non banale.

p.e., non Arizona e Colorado



3-colorabilità di grafi planari è NP-completo

Teorema. PLANAR-3-COLOR è **NP**-completo.

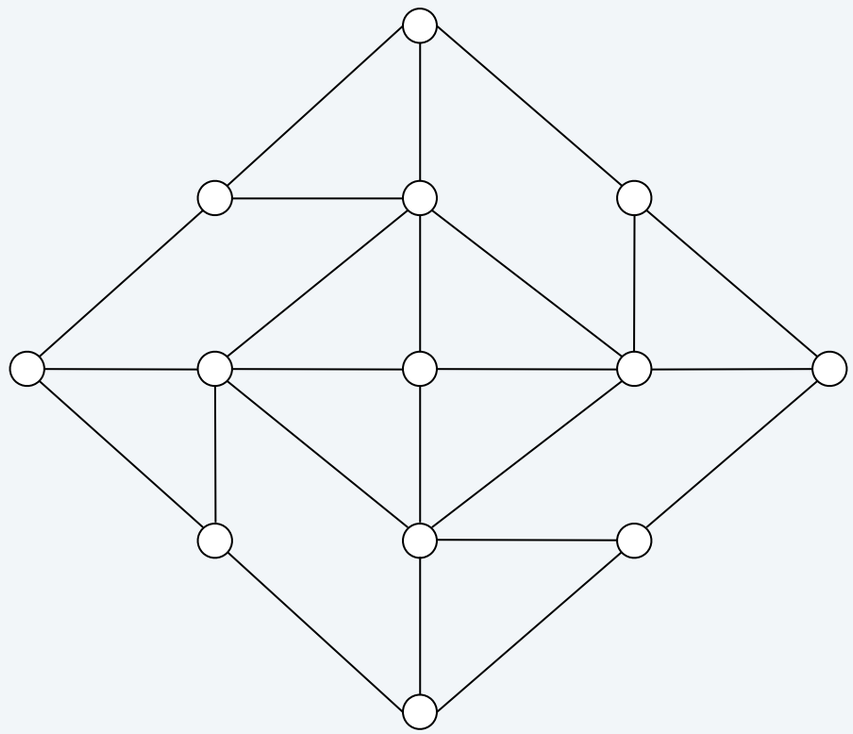
Dim.

- Facile vedere che PLANAR-3-COLOR $\in$ **NP**.
- Mostriamo che 3-COLOR $\leq_P$ PLANAR-3-COLOR.
- Data un'istanza di 3-COLOR G, costruiamo un'istanza di PLANAR-3-COLOR che è 3-colorabile sse G è 3-colorabile.

3-colorabilità di grafi planari è NP-completo

Lemma. Il grafo W (raffigurato qui sotto) è un grafo planare tale che:

- In qualunque 3-colorazione di W, vertici esterni opposti hanno lo stesso colore.
- Qualunque assegnazione di 3 colori ai vertici esterni in cui vertici opposti hanno lo stesso colore può essere estesa ad una 3-colorazione di W.



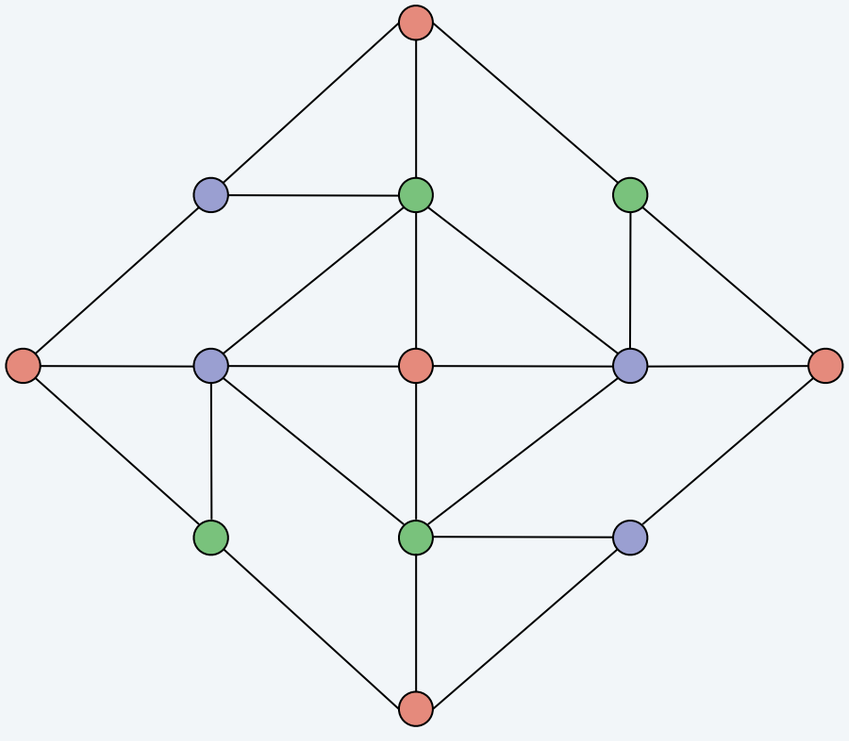
gadget planare W

3-colorabilità di grafi planari è NP-completo

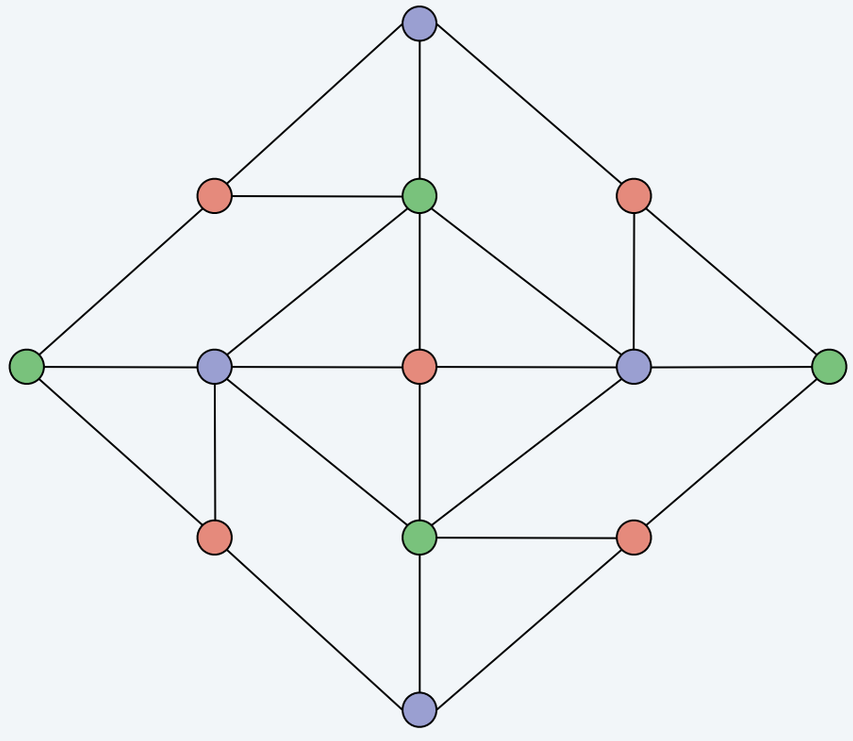
Lemma. Il grafo W (raffigurato qui sotto) è un grafo planare tale che:

- In qualunque 3-colorazione di W, vertici esterni opposti hanno lo stesso colore.
- Qualunque assegnazione di 3 colori ai vertici esterni in cui vertici opposti hanno lo stesso colore può essere estesa ad una 3-colorazione di W.

Dim. Le uniche 3-colorazioni (a meno di permutazioni) di W sono mostrate qui sotto. ▪



gadget planare W

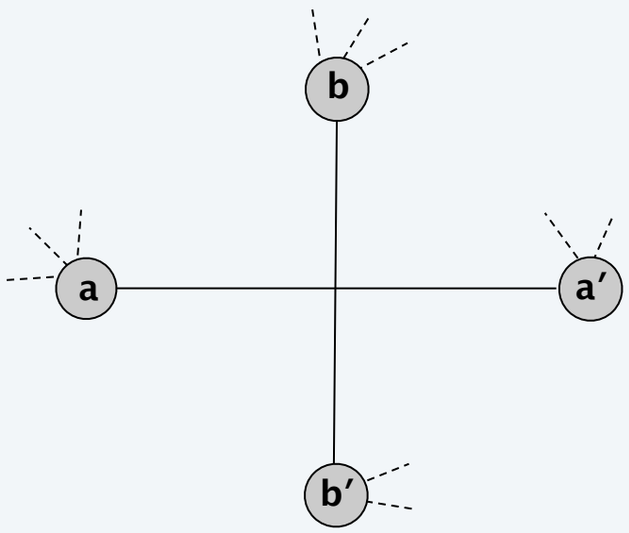


3-colorabilità di grafi planari è NP-completo

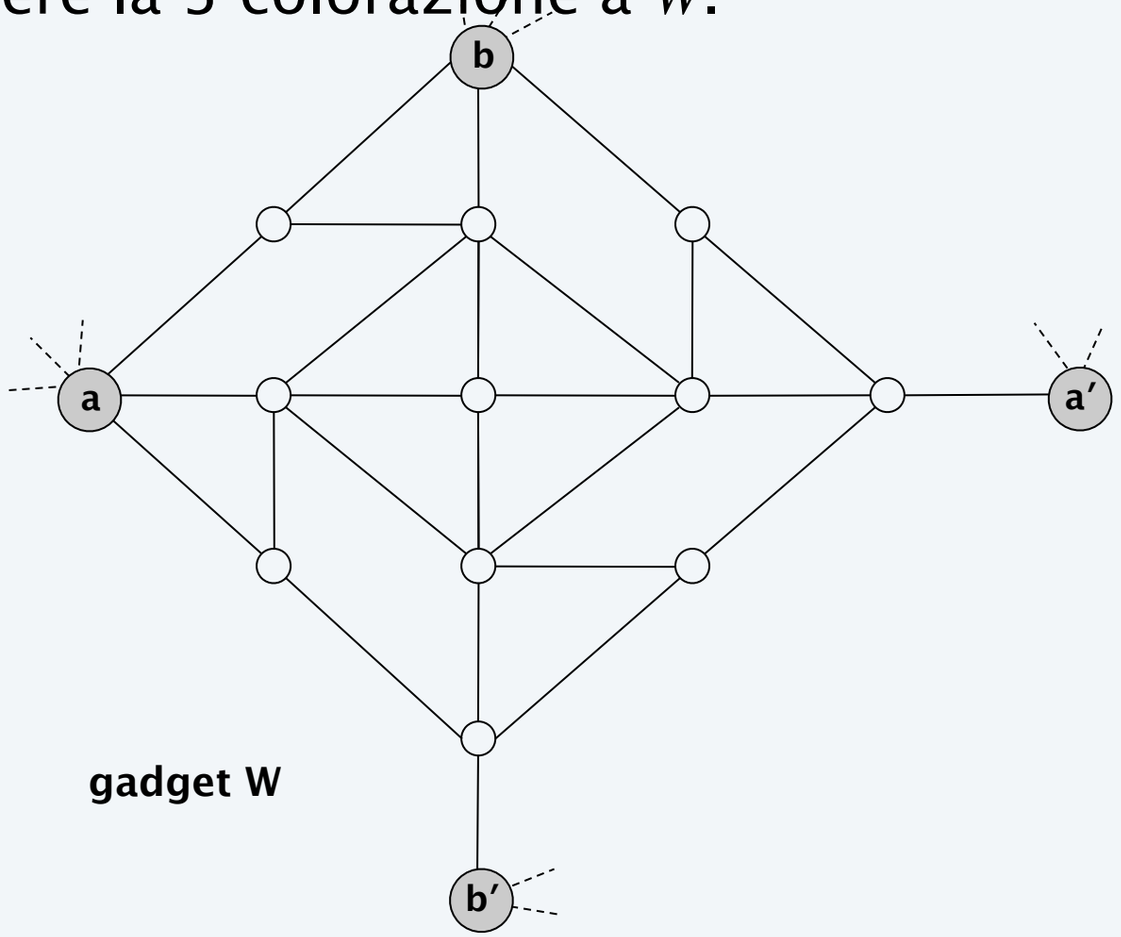
Costruzione. Data un'istanza G di 3-COLOR, immergi G nel piano, permettendo incroci. Forma un grafo planare G' rimpiazzando ciascun incrocio di archi con una copia del gadget planare W.

Lemma. G è 3-colorabile sse G' è 3-colorabile.

- In ogni 3-colorazione di W, $a \neq a'$ e $b \neq b'$.
- Se $a \neq a'$ e $b \neq b'$ allora si può estendere la 3-colorazione a W.



un incrocio



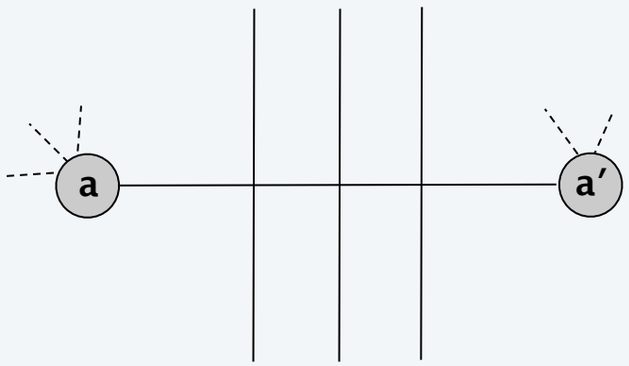
gadget W

3-colorabilità di grafi planari è NP-completo

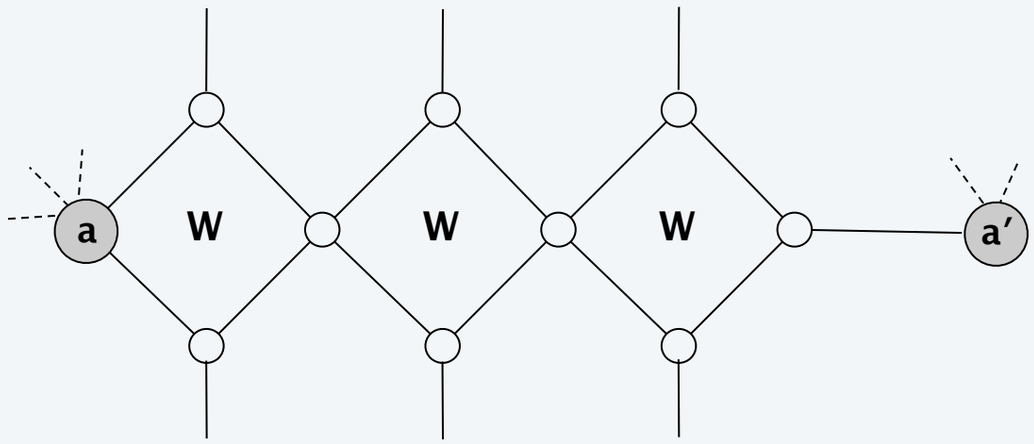
Costruzione. Data un'istanza G di 3-COLOR, immergi G nel piano, permettendo incroci. Forma un grafo planare G' rimpiazzando ciascun incrocio di archi con una copia del gadget planare W.

Lemma. G è 3-colorabile sse G' è 3-colorabile.

- In ogni 3-colorazione di W, $a \neq a'$ e $b \neq b'$.
- Se $a \neq a'$ e $b \neq b'$ allora si può estendere la 3-colorazione a W.



incroci multipli

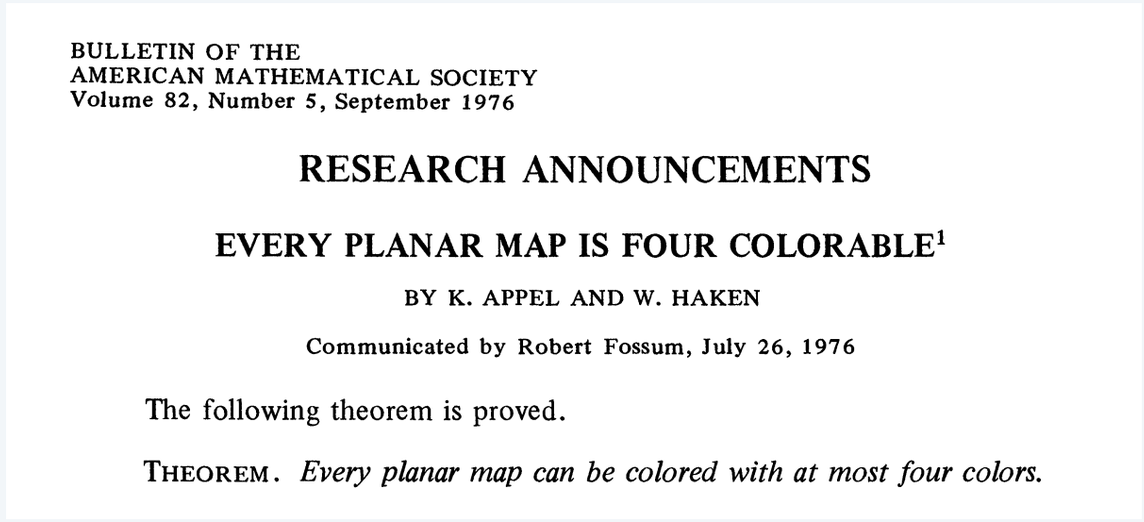


copie concatenate del grafo W

k-Colorabilità di mappe planari

Teorema. [Appel–Haken 1976] Ogni mappa planare è 4-colorabile.

- Ha risposto ad un problema vecchio di un secolo.
- Ha richiesto 50 giorni di calcoli per considerare molti casi particolari.
- Primo teorema importante dimostrato con l'ausilio del calcolatore.



BULLETIN OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 82, Number 5, September 1976

RESEARCH ANNOUNCEMENTS

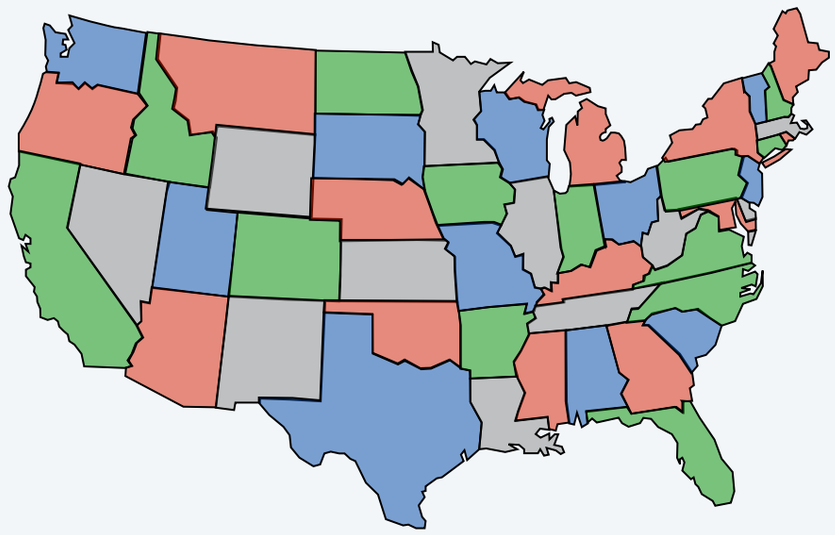
EVERY PLANAR MAP IS FOUR COLORABLE[1]

BY K. APPEL AND W. HAKEN

Communicated by Robert Fossum, July 26, 1976

The following theorem is proved.

THEOREM. *Every planar map can be colored with at most four colors.*



Note.

- Appel–Haken dà un algoritmo $O(n^4)$ per 4-colorare una mappa planare.
- Miglior risultato noto: $O(n^2)$ per 4-colorare; $O(n)$ per 5-colorare.
- Determinare se 3 colori sono sufficienti è un problema **NP**-completo.

Casi particolari polinomiali di problemi NP-ardui

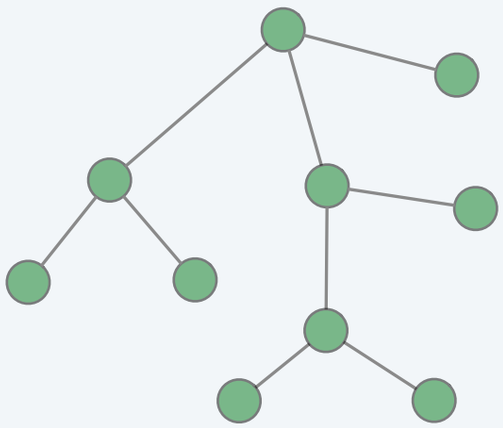
Alberi. VERTEX-COVER, INDEPENDENT-SET, LONGEST-PATH, GRAPH-ISOMORPHISM, ...

Grafi bipartiti. VERTEX-COVER, INDEPENDENT-SET, 3-COLOR, EDGE-COLOR, ...

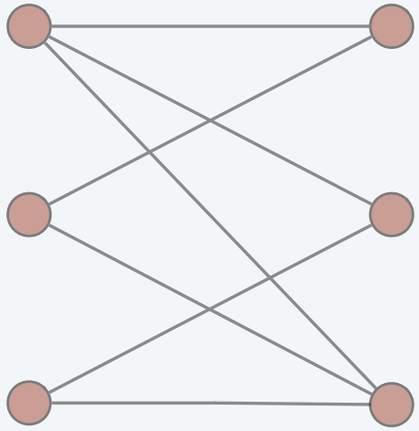
Grafi planari. MAX-CUT, ISING, CLIQUE, GRAPH-ISOMORPHISM, 4-COLOR, ...

Treewidth limitata. HAM-CYCLE, INDEPENDENT-SET, GRAPH-ISOMORPHISM, ...

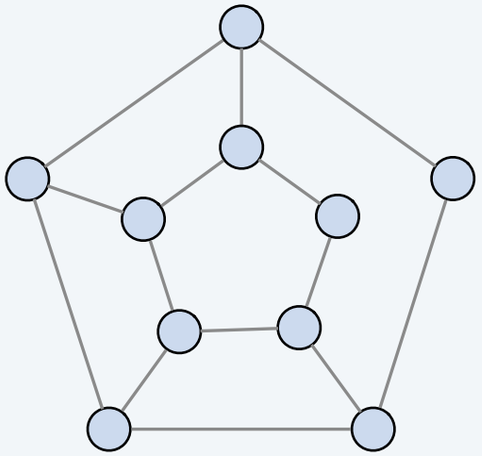
Interi piccoli. SUBSET-SUM, KNAPSACK, PARTITION, ...



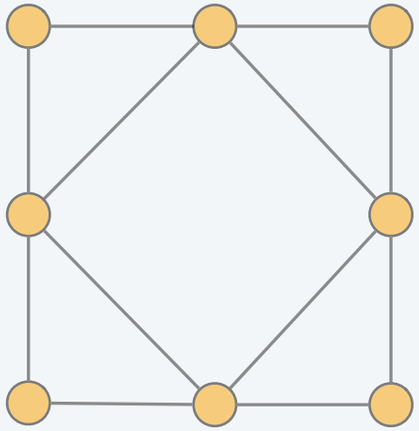
albero



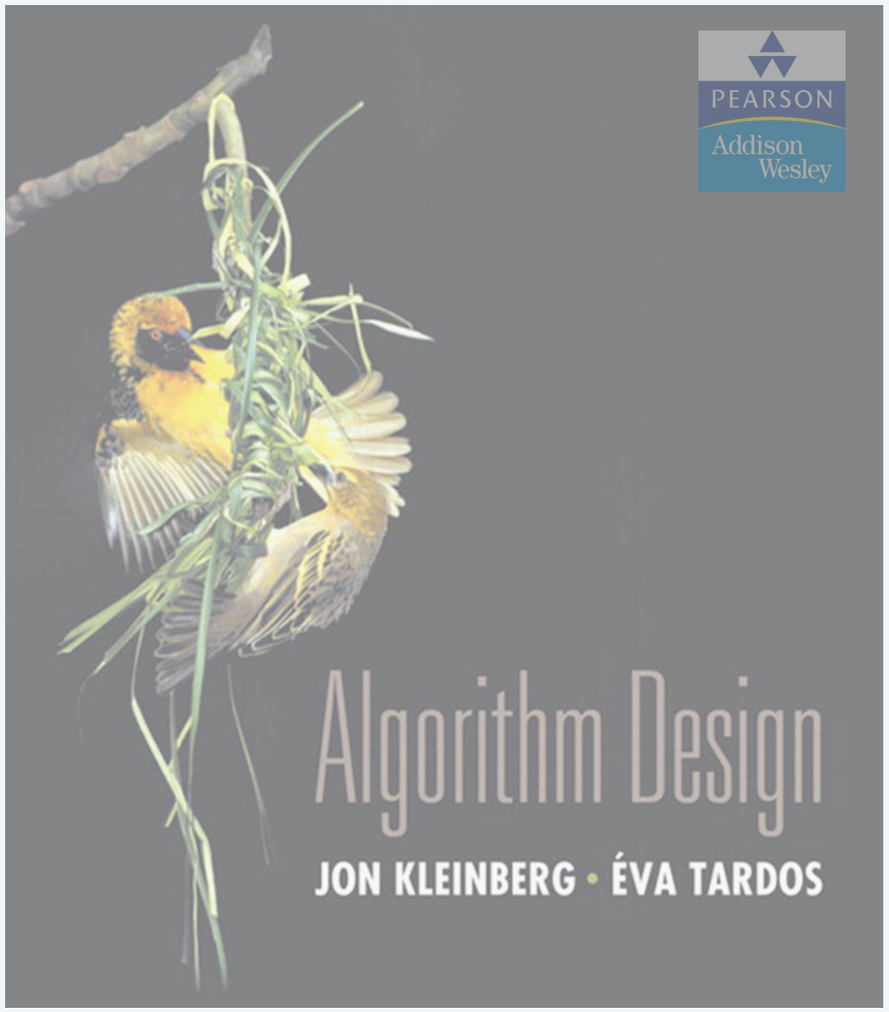
grafo bipartito



grafo planare



treewidth limitata



SECTION 11.8

INTRATTABILITÀ III

- *special cases: trees*
- *special cases: planarity*
- *algoritmi approssimanti: vertex cover*
- *approximation algorithms: knapsack*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

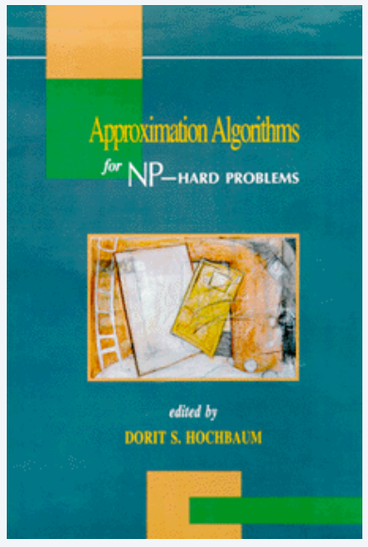
Algoritmi di approssimazione (o algoritmi approssimanti)

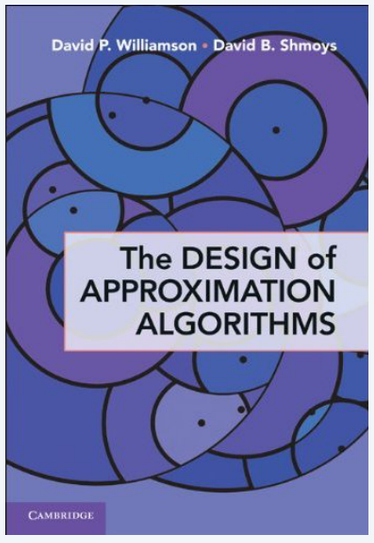
Algoritmo di ρ-approssimazione.

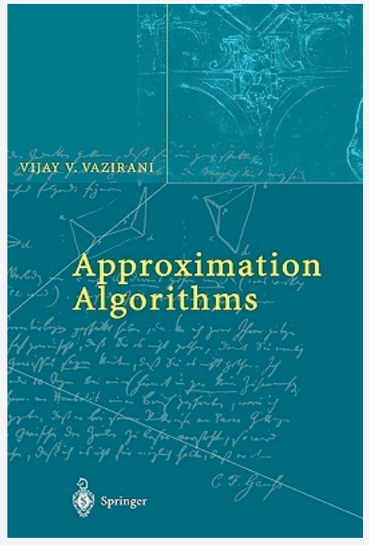
- Esegue in tempo polinomiale.
- Si applica ad istanze arbitrarie del problema.
- Garantisce di trovare una soluzione di valore entro un fattore ρ dal vero valore ottimo.

Es. Dato un grafo G, si può trovare un vertex cover che usa $\leq 2\,OPT(G)$ vertici in tempo $O(m+n)$.

Difficoltà. Occorre dimostrare che il valore della soluzione è vicino a quello dell'ottimo, senza di fatto conoscere il valore ottimo!



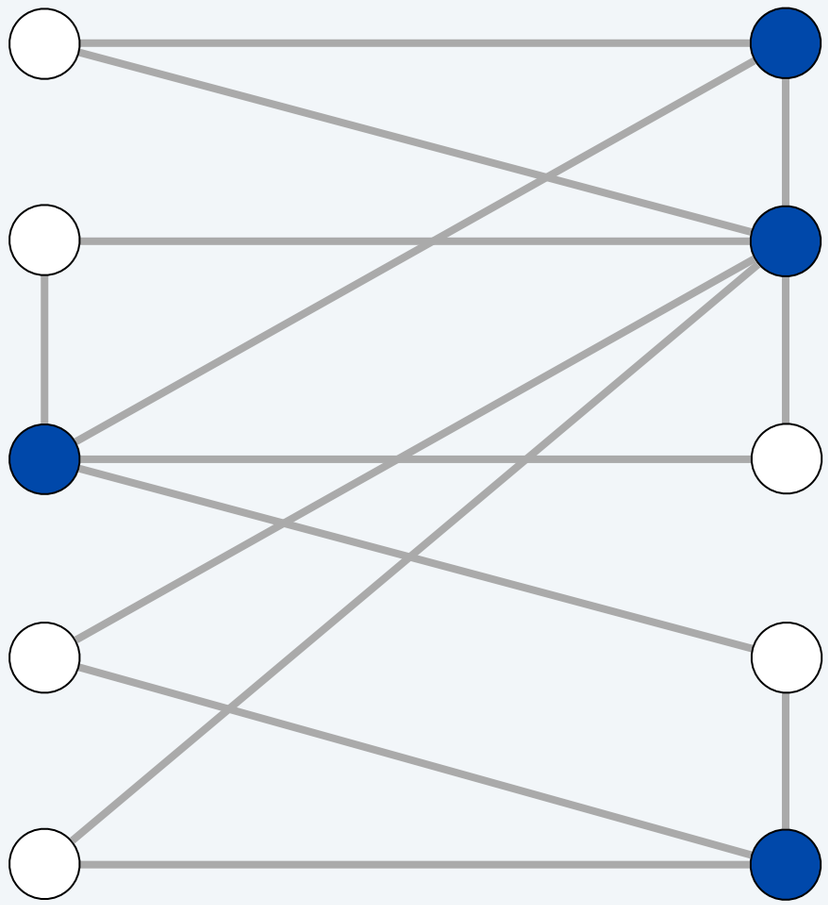




Vertex cover

VERTEX-COVER. Dato un grafo $G = (V, E)$, trova un vertex cover di taglia minima.

per ogni arco $(u, v) \in E$:
o $u \in S$, o $v \in S$, o entrambe



vertex cover di taglia 4

Vertex cover: algoritmo avido

VERTEX-COVER. Dato un grafo $G = (V, E)$, trova un vertex cover di taglia minima.

GREEDY-VERTEX-COVER(G)

$S \leftarrow \emptyset$.

$E' \leftarrow E$.

WHILE ($E' \neq \emptyset$)

  Sia $(u, v) \in E'$ un arco arbitrario.

  $M \leftarrow M \cup \{(u, v)\}$. ← M è un abbinamento

  $S \leftarrow S \cup \{u\} \cup \{v\}$.

  Cancella da E' tutti gli archi incidenti ad u o a v.

RETURN S.

ogni vertex cover deve includere almeno uno di questi; li prendiamo entrambi

Tempo di esecuzione. Può essere implementato con tempo $O(m + n)$.

Intrattabilità III: quiz 3

Dato un grafo G, sia M un qualunque abbinamento e sia S un qualunque vertex cover. Quale delle seguenti è sempre vera?

A. $|M| \leq |S|$

B. $|S| \leq |M|$

C. $|S| = |M|$

D. Nessuna delle precedenti.

Vertex cover: l'algoritmo avido è un algoritmo 2-approssimante

Teorema. Sia S^* un vertex cover di taglia minima. L'algoritmo avido trova un vertex cover S con $|S| \leq 2\,|S^*|$. ← l'algoritmo è 2-approssimante

Dim.

- S è un vertex cover. ← rimuoviamo archi solo se sono già coperti
- M è un abbinamento. ← quando (u, v) è aggiunto ad M, gli archi incidenti a u e v vengono rimossi
- $|S| = 2\,|M| \leq 2\,|S^*|$. ▪

  ↑ per costruzione ↑ dualità debole

Corollario. Sia M^* un abbinamento massimo. L'algoritmo avido trova un abbinamento M tale che $|M| \geq \frac{1}{2}\,|M^*|$.

Dim. $|M| = \frac{1}{2}\,|S| \geq \frac{1}{2}\,|M^*|$. ▪

↑ dualità debole

Inapprossimabilità del vertex cover

Teorema. [Dinur–Safra 2004] Se **P** ≠ **NP**, non esistono algoritmi ρ-approssimanti tempo-polinomiali per VERTEX-COVER con $\rho < 1.3606$.

On the Hardness of Approximating Minimum Vertex Cover

Irit Dinur* Samuel Safra†

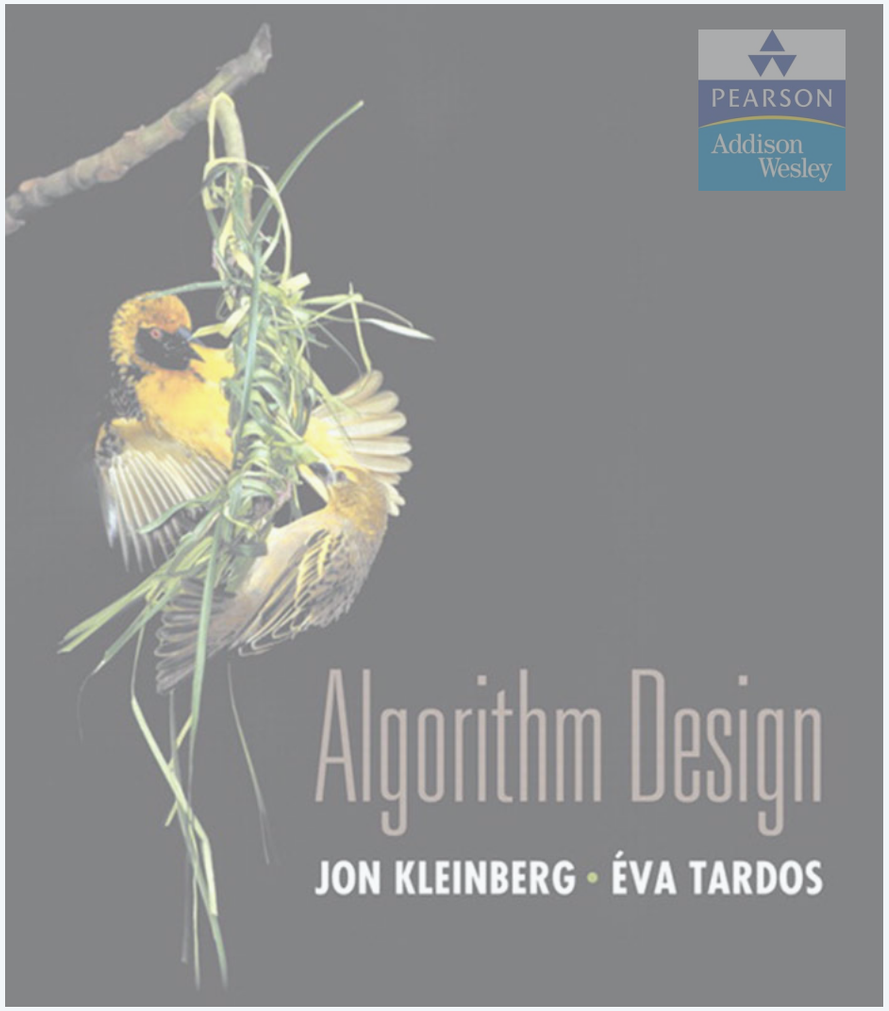
May 26, 2004

Abstract

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.

Problema aperto di ricerca. Colmare il gap tra 1.3606 e 2.

Congettura. Non c'è ρ-approssimazione per VERTEX-COVER con $\rho < 2$.



SECTION 11.8

INTRATTABILITÀ III

- *special cases: trees*
- *special cases: planarity*
- *approximation algorithms: vertex cover*
- *algoritmi approssimanti: bisaccia*
- *exponential algorithms: 3-SAT*
- *exponential algorithms: TSP*

Problema della bisaccia

Knapsack (variante di ottimizzazione).

- Dati n oggetti e una bisaccia.
- Oggetto i ha valore $v_i > 0$ e peso $w_i > 0$. ← assumiamo $w_i \leq W$ per ogni i
- La bisaccia ha limite di peso W.
- Scopo: riempire la bisaccia massimizzando il valore totale degli oggetti.

Es: $\{ 3, 4 \}$ ha valore 40.

oggetto	valore	peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

istanza originale (W = 11)

Knapsack è NP-completo

SUBSET-SUM. Dato un insieme X, valori $u_i \geq 0$, ed un intero U, esiste un sottoinsieme $S \subseteq X$ di elementi aventi somma esattamente U?

KNAPSACK (VARIANTE DI DECISIONE). Dato un insieme X, pesi $w_i \geq 0$, valori $v_i \geq 0$, un limite di peso W, ed un valore bersaglio V, esiste un sottoinsieme $S \subseteq X$ tale che:

$$\sum_{i \in S} w_i \;\leq\; W$$

$$\sum_{i \in S} v_i \;\leq\; V$$

Teorema. SUBSET-SUM $\leq_{\mathrm{P}}$ KNAPSACK.

Dim. Data l'istanza $(u_1, \ldots, u_n, U)$ di SUBSET-SUM, crea l'istanza KNAPSACK:

$$v_i = w_i = u_i$$

$$V = W = U$$

$$\sum_{i \in S} u_i \;\leq\; U$$

$$\sum_{i \in S} u_i \;\geq\; U$$

Knapsack: algoritmo di programmazione dinamica I

Def. $OPT(i, w)$ = massimo valore di un sottoinsieme degli oggetti $1,\ldots, i$ con limite superiore di peso w.

Caso 1. OPT non seleziona l'oggetto i.

- OPT sceglie al meglio tra $1, \ldots, i-1$ con limite di peso w.

Caso 2. OPT seleziona l'oggetto i.

- Nuovo limite di peso $= w - w_i$.
- OPT sceglie al meglio tra $1, \ldots, i-1$ con limite di peso $w - w_i$.

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{\, OPT(i-1,w),\ \ v_i + OPT(i-1, w-w_i)\,\} & \text{otherwise} \end{cases}$$

Teorema. Calcola il valore ottimo in tempo $O(n\,W)$.

- Non è polinomiale nella taglia dell'input.
- È polinomiale nella taglia dell'input se i pesi sono interi piccoli.

Knapsack: algoritmo di programmazione dinamica II

Def. $OPT(i, v)$ = minimo peso di un sottoinsieme degli oggetti $1,..., i$ con limite inferiore di valore v.

Nota. Il valore ottimo è il massimo v tale che $OPT(n, v) \leq W$.

Caso 1. OPT non seleziona l'oggetto i.

- OPT sceglie al meglio tra $1, \ldots, i-1$ in modo di avere valore ≥ v.

Case 2. OPT seleziona l'oggetto i.

- Contribuisce peso w_i, richiede di avere valore ulteriore $\geq v - v_i$.
- OPT sceglie al meglio tra $1, \ldots, i-1$ in modo di avere valore ≥ $v - v_i$.

$$OPT(i, v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min \{OPT(i-1, v),\ w_i + OPT(i-1, v - v_i)\} & \text{otherwise} \end{cases}$$

Knapsack: algoritmo di programmazione dinamica II

Teorema. L'algoritmo di programmazione dinamica II calcola il valore ottimo in tempo $O(n\,V)$, dove V è la somma dei valori degli oggetti.

Dim.

- Il valore ottimo della bisaccia soddisfa $V^* \leq V$.
- C'è un sottoproblema per ogni combinazione di oggetto e valore $v \leq V^*$.
- Ogni sottoproblema richiede tempo $O(1)$. ▪

Nota 1. Non è polinomiale nella taglia dell'input!

Nota 2. Il tempo è polinomiale nella taglia dell'input se i valori sono interi piccoli.

Corollario. L'algoritmo di programmazione dinamica II calcola il valore ottimo in tempo $O(n^2\,v_{\max})$, dove $v_{\max}$ è il massimo dei valori degli oggetti.

Dim. $V \leq n\,v_{\max}$.

Knapsack: schema di approssimazione tempo-polinomiale

Intuizione per l'algoritmo di approssimazione.

- Scala (arrotondandoli) tutti i valori per riportarli entro un range piccolo.
- Esegui l'algoritmo di programmazione dinamica II sull'istanza arrotondata.
- Restituisci gli oggetti ottimi dell'istanza arrotondata.

ogg.	valore	peso
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

istanza originale (W = 11)

ogg.	valore	peso
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

istanza arrotondata (W = 11)

Knapsack: schema di approssimazione tempo-polinomiale

Arrotonda tutti i valori:

- $0 < \varepsilon \leq 1$ = un parametro di precisione.
- $v_{\max}$ = valore più alto nell'istanza originale;
- θ = fattore di scala = $\varepsilon\, v_{\max} / 2n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta\,, \quad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Osservazione. Soluzioni ottime al problema con $\bar{v}$ sono equivalenti a soluzioni ottime al problema con $\hat{v}$.

Intuizione. $\bar{v}$ vicino a v, quindi la soluzione che usa $\bar{v}$ è quasi ottima; i valori $\hat{v}$ sono interi piccoli e quindi l'algoritmo di programmazione dinamica II è efficiente.

Knapsack: schema di approssimazione tempo-polinomiale

Teorema. Se S è una soluzione trovata dall'algoritmo di approssimazione e S^* è una soluzione valida,

$$(1+\epsilon)\sum_{i\in S} v_i \;\geq\; \sum_{i\in S^*} v_i$$

Dim. Sia S^* una soluzione valida (cioè che soddisfa il limite di peso).

$$\sum_{i\in S^*} v_i \;\leq\; \sum_{i\in S^*} \bar{v}_i$$

l'arrotondamento è verso l'alto

$$\leq\; \sum_{i\in S} \bar{v}_i$$

l'istanza arrotondata è risolta in modo ottimo da S

$$\leq\; \sum_{i\in S} (v_i + \theta)$$

valore arrotondato è entro θ dal valore originale

$$\leq\; \sum_{i\in S} v_i \;+\; n\theta$$

$|S| \leq n$

$$=\; \sum_{i\in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{\max}$$

$\theta = \varepsilon\, v_{\max} / 2n$

$$\leq\; (1+\epsilon)\sum_{i\in S} v_i$$

$v_{\max} \leq 2\,\Sigma_{i\in S}\, v_i$

sottoinsieme contenente solo l'oggetto di valore massimo

nel caso particolare $S^* = \{ \max \}$

$$v_{\max} \;\leq\; \sum_{i\in S} v_i \;+\; \tfrac{1}{2}\,\epsilon\, v_{\max}$$

$$\leq\; \sum_{i\in S} v_i \;+\; \tfrac{1}{2}\, v_{\max}$$

quindi

$$v_{\max} \;\leq\; 2\sum_{i\in S} v_i$$

Knapsack: schema di approssimazione tempo-polinomiale

Teorema. Per ogni $\varepsilon > 0$, l'algoritmo di arrotondamento calcola una soluzione valida il cui valore è entro un fattore $(1 + \varepsilon)$ dall'ottimo, in tempo $O(n^3 / \varepsilon)$.

Dim.

- Già dimostrato che il valore è entro un fattore $(1 + \varepsilon)$ dall'ottimo.
- Il tempo di esecuzione dell'algoritmo è $O(n^2 \hat{v}_{\max})$, dove

$$\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{2n}{\epsilon} \right\rceil$$