Algoritmi Euristici

Corso di Laurea in Informatica e Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni: Lunedì 13.30 - 15.30 in Aula G30
Giovedì 13.30 - 15.30 in Aula G30
Ricevimento: su appuntamento
Tel.: 02 503 16235
E-mail: roberto.cordone@unimi.it
Web page: http://homes.di.unimi.it/~cordone/courses/2018-ae/2018-ae.html

Problemi e algoritmi

Un problema è una domanda su un sistema di oggetti matematici
In genere, si può porre la stessa domanda su molti sistemi simili

- istanza $I \in \mathcal{I}$ è ciascuno specifico sistema oggetto della domanda (per es., "*n è primo?*" è un problema, "*7 è primo?*" una sua istanza)
- soluzione $S \in \mathcal{S}$ è la risposta che corrisponde a ogni istanza

Formalmente, un problema è la funzione che lega istanze e soluzioni

$$P : \mathcal{I} \to \mathcal{S}$$

A priori, non sappiamo calcolarla!

Algoritmo: una procedura formale, deterministica, fatta di passi elementari, in sequenza finita

Un algoritmo per un problema P è un algoritmo i cui passi sono determinati da un'istanza $I \in \mathcal{I}$ e producono una soluzione $S \in \mathcal{S}$

$$A : \mathcal{I} \to \mathcal{S}$$

Un algoritmo non solo definisce una funzione, ma anche la calcola

Se la funzione è la stessa, l'algoritmo è esatto; altrimenti, euristico

Costo di un algoritmo euristico

Un algoritmo euristico è utile se è

1. efficiente: "costa" molto meno di un algoritmo esatto
2. efficace: fornisce "spesso" una soluzione "vicina" a quella corretta

Cominciamo discutendo l'efficienza

Con "costo" di un algoritmo (esatto o euristico) si intende

- non il costo monetario di acquistarlo o realizzarlo
- ma il costo computazionale di eseguirlo
  - spazio occupato in memoria
  - tempo richiesto per terminare

Si discute molto più spesso il tempo, perché

- lo spazio è una risorsa riutilizzabile, il tempo no
- per consumare spazio, si usa implicitamente altrettanto tempo
- è tecnicamente più facile distribuire l'uso di spazio che di tempo

Spazio e tempo sono in parte intercambiabili:
si può ridurre il consumo dell'uno aumentando quello dell'altro

Una misura utile del tempo

Il tempo necessario a risolvere un problema dipende da molti aspetti

- l'istanza specifica da risolvere
- l'algoritmo usato
- la macchina che esegue l'algoritmo
- ...

Vogliamo arrivare a una misura del tempo di calcolo che sia

- slegata dalla tecnologia, cioè uguale per molte macchine diverse
- sintetica, cioè riassumibile in una semplice espressione simbolica
- ordinale, cioè che consenta di stabilire se un algoritmo è più efficiente di un altro

Il tempo di calcolo in secondi per ogni istanza non soddisfa questi requisiti

Complessità temporale

La complessità asintotica di un algoritmo nel caso pessimo fornisce una misura del tempo di calcolo dell'algoritmo attraverso i seguenti passaggi

1. misuriamo il tempo col numero T di operazioni elementari eseguite (per ottenere una misura indipendente dallo specifico computer)
2. scegliamo un valore n che misuri la dimensione di un'istanza (per es., il numero di elementi dell'insieme base, di variabili o formule della CNF, di righe o colonne della matrice, di nodi o archi del grafo)
3. troviamo il tempo di calcolo massimo per le istanze di dimensione n

$$T(n) = \max_{I \in \mathcal{I}_n} T(I) \qquad n \in \mathbb{N}$$

(questo riduce la complessità a una funzione $T : \mathbb{N} \to \mathbb{N}$)

4. approssimiamo $T(n)$ per eccesso e/o difetto con una funzione $f(n)$ più semplice, di cui interessa solo l'andamento per $n \to +\infty$ (è più importante che l'algoritmo sia efficiente su dimensioni grandi)
5. raccogliamo le funzioni in classi con la stessa approssimante (la relazione di approssimazione è una relazione di equivalenza)

Notazione Θ

$$T(n) \in \Theta(f(n))$$

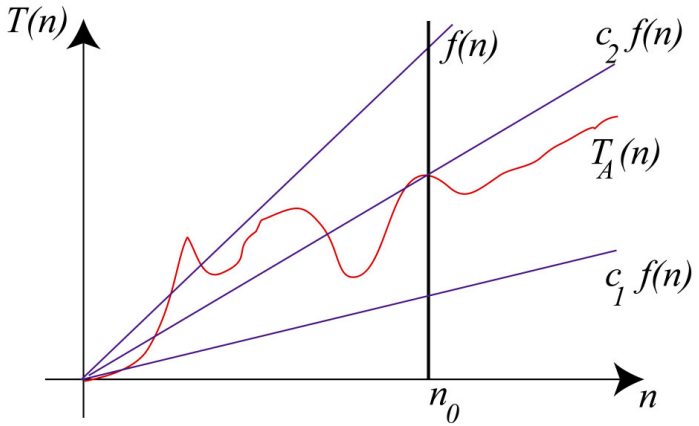
significa formalmente che

$$\exists c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N} : c_1\, f(n) \leq T(n) \leq c_2\, f(n) \text{ for all } n \geq n_0$$

dove c_1, c_2 e n_0 sono indipendenti da n

$T(n)$ è *"chiusa a sandwich"* fra $c_1 f(n)$ e $c_2 f(n)$

- per qualche valore *"piccolo"* di c_1
- per qualche valore *"grande"* di c_2
- per ogni valore *"grande* di n
- per qualche definizione di *"piccolo"* e *"grande"*



Asintoticamente, $f(n)$ stima $T(n)$ a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è almeno e al massimo proporzionale ai valori della funzione $f(n)$

Notazione O

$$T(n) \in O(f(n))$$

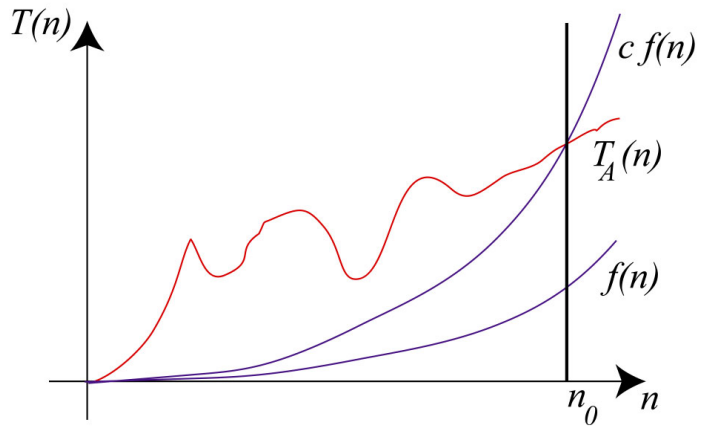
significa formalmente che

$$\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N} : T(n) \leq c\, f(n) \text{ for all } n \geq n_0$$

dove c, e n_0 sono indipendenti da n

$T(n)$ è *"dominata"* da $cf(n)$

- per qualche valore *"grande"* di c
- per ogni valore *"grande* di n
- per qualche definizione di *"grande"*



Asintoticamente, $f(n)$ stima $T(n)$ per eccesso a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è al massimo proporzionale ai valori della funzione $f(n)$

Notazione Ω

$$T(n) \in \Omega(f(n))$$

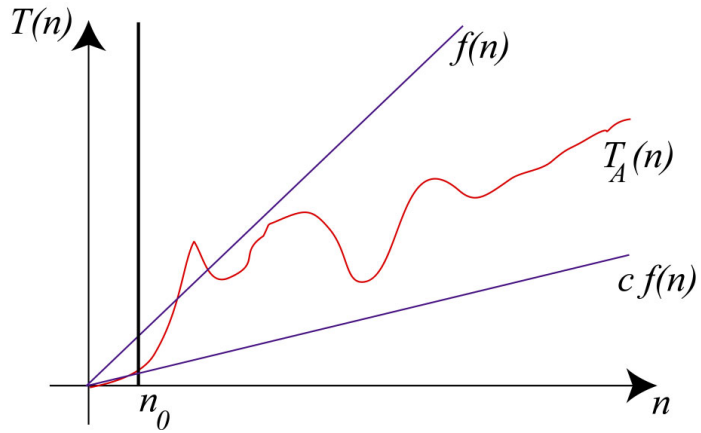
significa formalmente che

$$\exists c > 0, n_0 \in \mathbb{N} : T(n) \geq c\, f(n) \text{ for all } n \geq n_0$$

dove c e n_0 sono indipendenti da n

$T(n)$ *"domina"* $cf(n)$

- per qualche valore *"piccolo"* di c
- per ogni valore *"grande* di n
- per qualche definizione di *"piccolo"* e *"grande"*



Asintoticamente, $f(n)$ stima $T(n)$ per difetto a meno di un fattore moltiplicativo:

- per istanze grandi, il tempo di calcolo è almeno proporzionale ai valori della funzione $f(n)$

L'algoritmo esaustivo

Concentriamoci sui problemi di Ottimizzazione Combinatoria

Definiamo la dimensione di un'istanza come cardinalità dell'insieme base

$$n = |B|$$

L'algoritmo esaustivo

- considera ogni sottoinsieme $x \subseteq B$, cioè ogni $x \in 2^B$
- valuta se è ammissibile ($x \in X$) in tempo $\alpha(n)$
- in caso positivo, valuta il valore dell'obiettivo $f(x)$ in tempo $\beta(n)$
- eventualmente aggiorna il miglior valore trovato sinora

Per quanto $\alpha(n)$ e $\beta(n)$ siano piccole, o si riesca a evitare $\beta(n)$, l'algoritmo esaustivo ha complessità almeno esponenziale

$$T(n) \in \Omega(2^n)$$

Di solito, $\alpha(n)$ e $\beta(n)$ sono polinomi e $T(n)$ risulta "solo" esponenziale

$$T(n) \in O(2^n(\alpha(n) + \beta(n)))$$

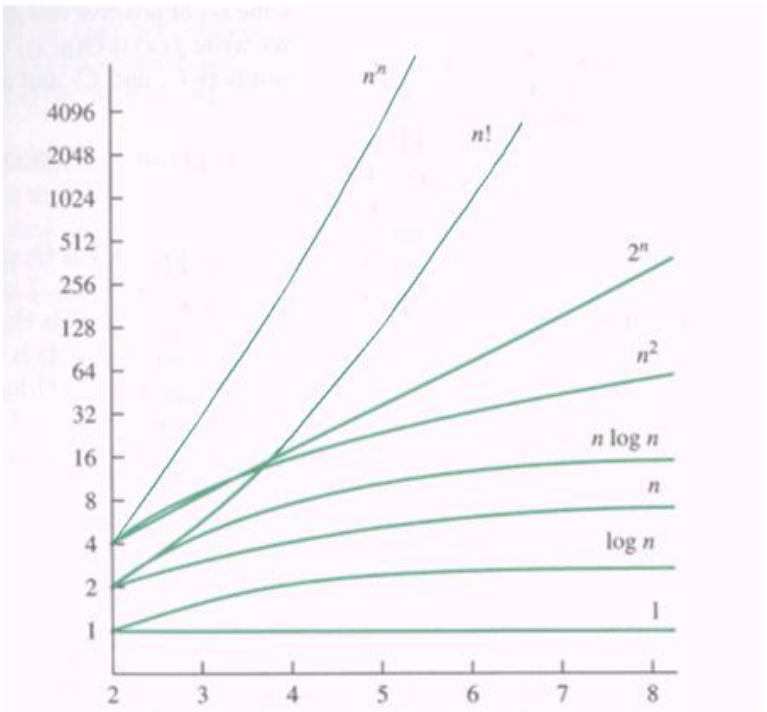
Complessità polinomiale ed esponenziale

In Ottimizzazione Combinatoria, la distinzione fondamentale è tra

- complessità polinomiale: $T(n) \in O(n^d)$ per una costante $d > 0$
- complessità esponenziale: $T(n) \in \Omega(d^n)$ per una costante $d > 1$

Gli algoritmi del primo tipo sono efficienti, quelli del secondo inefficienti

In genere, gli algoritmi euristici sono polinomiali e
sono utili per problemi i cui algoritmi esatti sono esponenziali



Assumendo 1 operation/μsec

n	n^2 op.	2^n op.
1	1μ sec	2μ sec
10	0.1 msec	1 msec
20	0.4 msec	1 sec
30	0.9 msec	17.9 min
40	1.6 msec	12.7 giorni
50	2.5 msec	35.7 anni
60	3.6 msec	366 secoli

Trasformazioni e riduzioni di problemi

Talvolta è possibile manipolare un problema P costruendo un problema Q e manipolare la soluzione di Q per ottenere quella di P

Trasformazione polinomiale $P \preceq Q$: data qualsiasi istanza di P

- si costruisce un'istanza di Q in tempo polinomiale
- si risolve l'istanza di Q con un algoritmo A
- dalla soluzione ottenuta, si costruisce la soluzione dell'istanza di P

Esempi: $VCP \preceq SCP$, $MCP \preceq MISP$ e $MISP \preceq MCP$

Riduzione polinomiale $P \preceq Q$: data qualsiasi istanza di P

- si esegue un numero polinomiale di volte un algoritmo A
- su istanze di Q costruite in tempo polinomiale dall'istanza di P e dai risultati delle chiamate precedenti
- dalle soluzioni ottenute, si costruisce la soluzione dell'istanza di P

Esempi: $BPP \preceq PMSP$ e $PMSP \preceq BPP$

In entrambi i casi

- se A è polinomiale/esponenziale, l'algoritmo complessivo è polinomiale/esponenziale
- se A è esatto/euristico, l'algoritmo complessivo è esatto/euristico

Problemi di ottimizzazione in forma di riconoscimento

Una riduzione polinomiale molto generale lega

- problema in forma di ottimizzazione: data una funzione f e una regione ammissibile X, qual è il minimo di f in X?

$$f^* = \min_{x \in X} f = ?$$

Questo è un classico problema di ottimizzazione

- problema in forma di riconoscimento: data una funzione f, un valore k e una regione ammissibile X, esistono soluzioni non peggiori di k?

$$\exists x \in X : f(x) \leq k?$$

Questo è un classico problema di decisione

Le due forme sono polinomialmente equivalenti:

- data la forma di riconoscimento, si ignora k, ottenendo la forma di ottimizzazione, si risolve e si paragona f^* con k:
  se $f^* \leq k$, la soluzione è *Sì*, altrimenti *No*
- data la forma di ottimizzazione, si ipotizza una soglia k e si risolve la forma di riconoscimento; si trova f^* per ricerca dicotomica su k:
  se la soluzione è *Sì*, si riduce k; se è *No*, si aumenta

Difetti della complessità nel caso pessimo

La complessità nel caso pessimo presenta un grande difetto:

- azzera l'informazione sulle istanze meno difficili
- fornisce una sovrastima brutale del tempo di calcolo, talvolta così lontana da essere inutile (anche se raramente)
  (ad es., l'*algoritmo del simplesso per la Programmazione Lineare*)

In pratica, le istanze difficili potrebbero essere rare o irrealistiche

Per ovviare si può studiare

- complessità parametrizzata, cioè esprimere T in funzione di qualche altro parametro rilevante k oltre che della dimensione n: $T(n, k)$
- complessità nel caso medio, cioè ipotizzare una distribuzione di probabilità su $\mathcal{I}$ e valutare il valore atteso di $T(I)$ su $\mathcal{I}_n$

$$T(n) = E[T(I) \,|\, I \in \mathcal{I}_n]$$

I due approcci si possono combinare se la distribuzione di probabilità ha un parametro k, ottenendo una complessità $T(n, k)$ nel caso medio

Complessità parametrizzata

Si cerca un altro parametro k, diverso dalla dimensione n, che abbia una forte influenza sul tempo di calcolo dell'algoritmo considerato

- il valore di qualche costante numerica importante (per es., la capacità nel *KP*, *CMSTP*..., il costo massimo per arco nel *TSP*,...)
- il numero di elementi non nulli nei problemi su matrici
- il numero massimo di letterali per formula nei problemi logici
- il grado massimo, il diametro, ecc... nei problemi su grafo
- la cardinalità della soluzione

Esistono algoritmi esponenziali in k e polinomiali in n, dunque

- efficienti su alcune istanze
- inefficienti su altre istanze

Se k è parte dell'istanza, sapremo a priori se l'algoritmo è efficiente

Se k fa parte della soluzione, lo sapremo solo a posteriori
(*ma è utile facendo ricerca binaria sulla forma di riconoscimento*)

Un esempio: il *VCP*

Algoritmo esaustivo: analizzare tutti i 2^n sottoinsiemi di vertici verificando almeno uno copre tutti i lati e ha cardinalità $\leq k$

$$T(n) \in \Theta\left(2^n (m+n)\right)$$

Ora consideriamo il *VCP* in forma di riconoscimento (k-*VCP*):
dato un grafo $G = (V, E)$ con $n = |V|$ e $m = |E|$ e un numero $k \leq n$, esiste un sottoinsieme di al massimo k vertici che coprono tutti i lati?

Risolvendo $\log_2 n$ forme di tal genere si risolve la forma di ottimizzazione

Algoritmo ingenuo: analizzare tutti i sottoinsiemi di k vertici verificando se almeno uno copre tutti i lati

$$T(n, k) \in \Theta\left(n^k m\right)$$

Per k fissato è polinomiale (*ma quasi sempre molto lento*)

Algoritmo 1: *Bounded tree search*

Proprietà utile: per ogni lato $(u, v) \in E$, qualsiasi soluzione ammissibile deve contenere almeno uno dei vertici: $x \cap (u, v) \neq \emptyset$

Algoritmo di *bounded tree search*:

1. scelto (u, v) qualsiasi, esploriamo due casi: $u \in x$ e $u \notin x$, $v \in x$
2. per ciascun caso aperto, cancelliamo i vertici di x e i lati coperti

$$V := V \setminus x \qquad E := E \setminus \{e \in E : e \cap x \neq \emptyset\}$$

(*I lati coperti da vertici in x non danno più problemi*)

3. se $|x| \leq k$ ed $E = \emptyset$, abbiamo trovato la soluzione cercata
4. se $|x| = k$ e $E \neq \emptyset$, questo caso non ha soluzioni ammissibili
5. altrimenti chiudiamo il caso e torniamo al passo 1

La complessità è $T(n, k) \in \Theta\left(2^k m\right)$, polinomiale in n (infatti $m \leq n^2/2$)

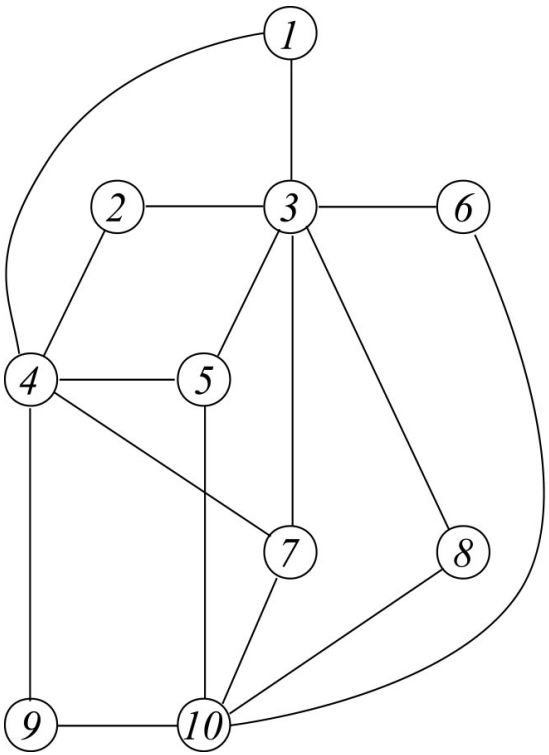
Per $n \gg 2$, questo algoritmo è molto più efficiente di quello ingenuo

Esempio 1: *Bounded tree search*

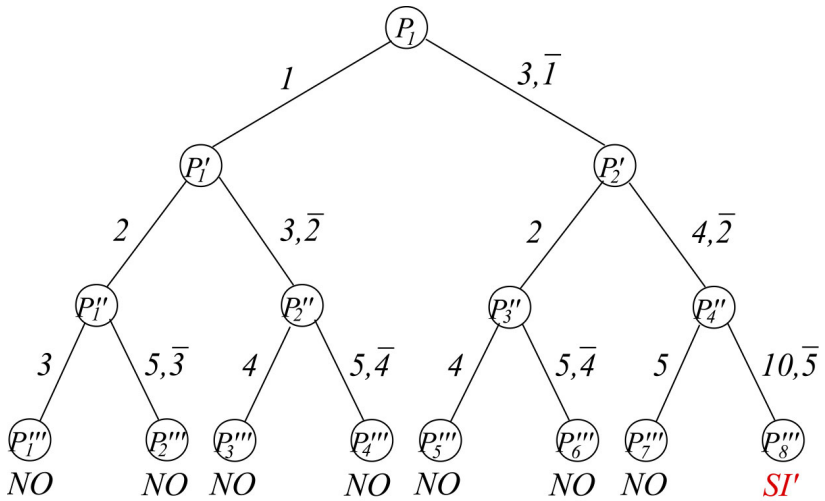
Dato il grafo seguente ($n = 10$, $m = 16$), esiste una soluzione con $k \leq 3$?

Algoritmo esaustivo: $T(n) \in \Theta(2^n(m+n))$, con $2^n(m+n) = 1\,024 \cdot 26$

Algoritmo ingenuo: $T(n,k) \in \Theta(n^k m)$, con $n^k m = 1\,000 \cdot 16$



Algoritmo di bounded tree search
(*scegliamo i lati in ordine lessicografico*)



$T(n,k) \in \Theta(2^k m)$, con $2^k m = 8 \cdot 16$

Esempio 2: *Kernelization*

Consiste nel ridurre l'istanza a una molto più piccola con ugual soluzione

Proprietà utile: ogni vertice v di grado $\delta_v \geq k+1$ deve appartenere a qualsiasi soluzione ammissibile di valore $\leq k$

Altrimenti, ognuno dei $k+1$ lati va coperto da un vertice diverso

Algoritmo di *kernelization*:

- partiamo con un sottoinsieme di vertici vuoto: $x := \emptyset$
- aggiungiamo alla soluzione tutti i vertici di grado $\geq k+1$

$$\delta_v \geq k+1 \;\Rightarrow\; x := x \cup \{v\}$$

- aggiorniamo k: $k := k - |x|$
- cancelliamo i vertici di grado nullo, quelli di x e i lati già coperti

$$V := \{v \in V : \delta_v > 0\} \setminus x \qquad E := \{e \in E : e \cap x \neq \emptyset\}$$

- se $|E| > k^2$, non ci sono soluzioni ammissibili (k *vertici non bastano*)
- se $|E| \leq k^2$, $|V| \leq 2k^2$; applichiamo l'algoritmo esaustivo

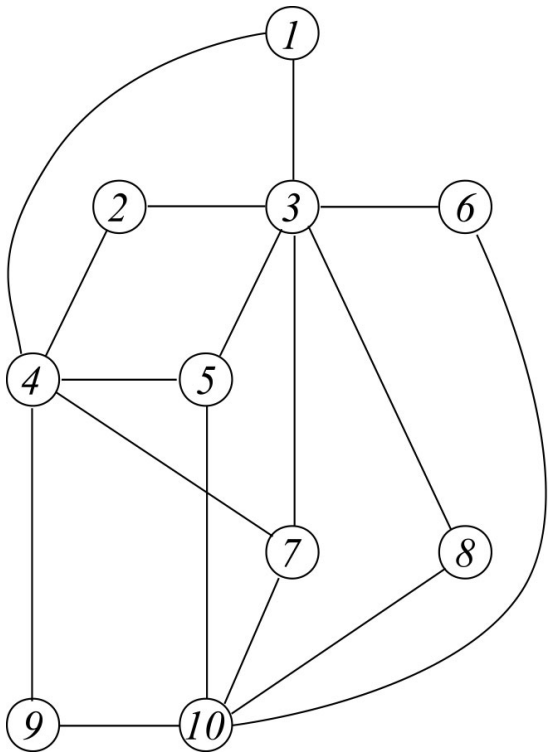
La complessità è $T(n,k) \in \Theta\left(n + m + 2^{2k^2} k^2\right)$

Esempio 2: *Kernelization*

Dato il grafo seguente ($n = 10$, $m = 16$), esiste una soluzione con $k \leq 3$?

Algoritmo esaustivo: $T(n) \in \Theta\left(2^n (m+n)\right)$, con $2^n (m+n) = 1\,024 \cdot 26$

Algoritmo ingenuo: $T(n,k) \in \Theta\left(n^k m\right)$, con $n^k m = 1\,000 \cdot 16$

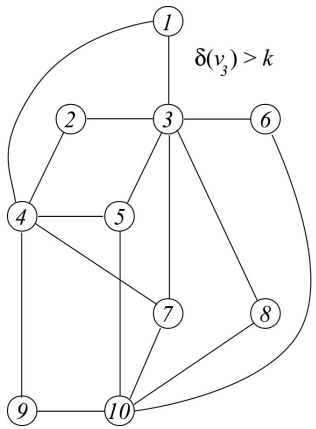


Algoritmo di kernelization

$T(n,k) \in \Theta\left(m + n + 2^{2k^2} k^2\right)$, con

$m + n + 2^{2k^2} k^2 = 26 + 2^{18} \cdot 9 \approx 2.4 \cdot 10^6$

Ma in realtà è risolto dopo $m + n$!

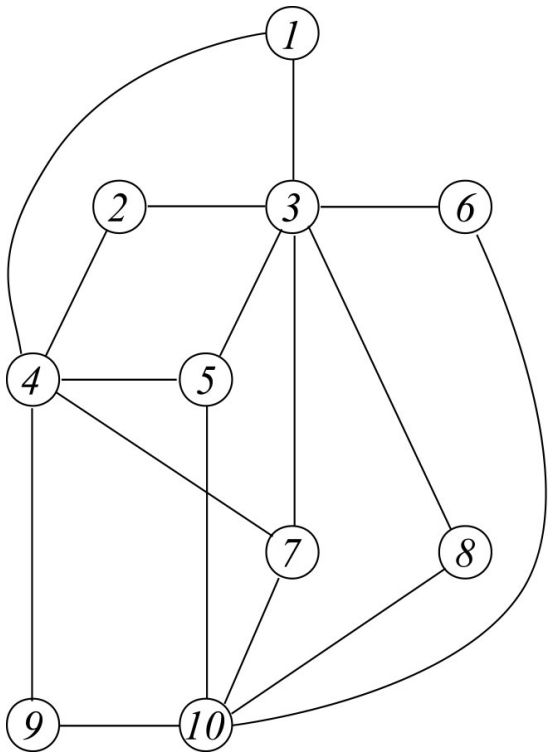


Esempio 2: *Kernelization*

Dato il grafo seguente ($n = 10$, $m = 16$), esiste una soluzione con $k \leq 3$?

Algoritmo esaustivo: $T(n) \in \Theta\left(2^n (m+n)\right)$, con $2^n (m+n) = 1\,024 \cdot 26$

Algoritmo ingenuo: $T(n,k) \in \Theta\left(n^k m\right)$, con $n^k m = 1\,000 \cdot 16$

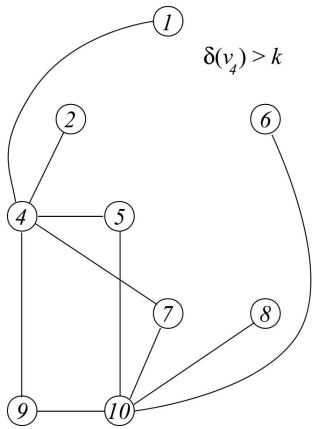


Algoritmo di kernelization

$T(n,k) \in \Theta\left(m + n + 2^{2k^2} k^2\right)$, con

$m + n + 2^{2k^2} k^2 = 26 + 2^{18} \cdot 9 \approx 2.4 \cdot 10^6$

Ma in realtà è risolto dopo $m + n$

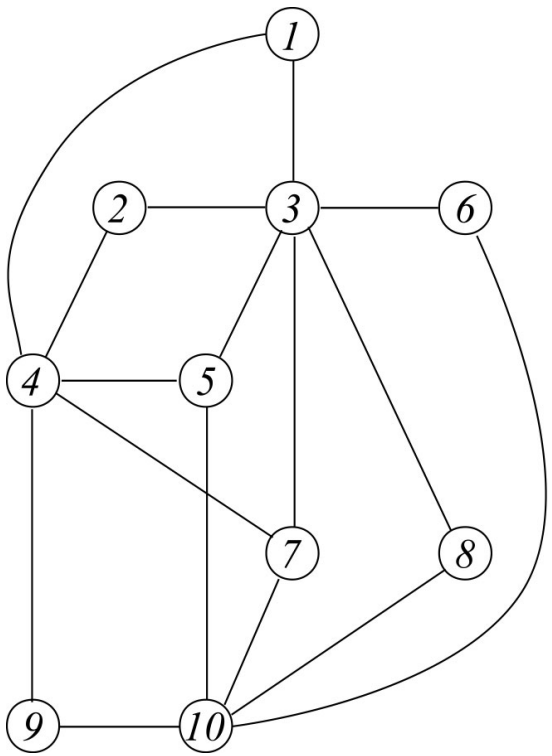


Esempio 2: *Kernelization*

Dato il grafo seguente ($n = 10$, $m = 16$), esiste una soluzione con $k \leq 3$?

Algoritmo esaustivo: $T(n) \in \Theta\left(2^n (m+n)\right)$, con $2^n (m+n) = 1\,024 \cdot 26$

Algoritmo ingenuo: $T(n,k) \in \Theta\left(n^k m\right)$, con $n^k m = 1\,000 \cdot 16$

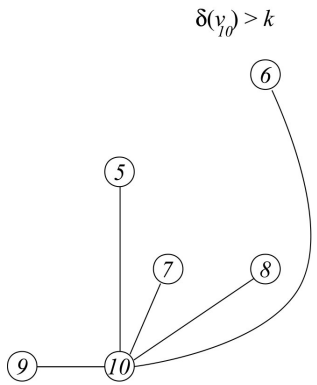


Algoritmo di kernelization

$T(n,k) \in \Theta\left(m + n + 2^{2k^2} k^2\right)$, con

$m + n + 2^{2k^2} k^2 = 26 + 2^{18} \cdot 9 \approx 2.4 \cdot 10^6$

Ma in realtà è risolto dopo $m + n$!



Complessità nel caso medio

Vi sono algoritmi efficienti su quasi tutte le istanze, inefficienti su poche
(*ad es., l'algoritmo del simplesso per la Programmazione Lineare*)

Idea: valutare il valore atteso di $T(I)$ su $\mathcal{I}_n$ per ogni $n \in \mathbb{N}$

$$T(n) = E[T(I) \,|\, I \in \mathcal{I}_n]$$

Questo richiede di definire la distribuzione di probabilità delle istanze

- l'ipotesi più frequente è quella della equiprobabilità
(*tipica ipotesi in caso di ignoranza assoluta*)
- altre ipotesi richiedono un modello probabilistico del problema
(*spesso dipendente da parametri*)

Istanze casuali: matrici

Che significa dare una probabilità a ogni istanza di un problema?

- per uno studio a priori delle prestazioni di un algoritmo, si ricorre a modelli semplici, trattabili algebricamente
- per uno studio a posteriori, cioè compiuto eseguendo l'algoritmo, volto a un'applicazione specifica, si ricorre a modelli che simulino l'applicazione

Consideriamo per ora i modelli semplici più comuni

Matrici casuali binarie con un dato numero di righe m e colonne n

1. modello a probabilità uniforme p:

$$Pr[a_{ij} = 1] = p \qquad (i = 1, \ldots, m;\ j = 1, \ldots, n)$$

   Se $p = 0.5$, coincide col modello a equiprobabilità di tutte le istanze

2. modello a densità fissata δ: dati gli mn elementi della matrice, se ne estraggono δmn casualmente in modo uniforme e si pongono a 1

   Ovviamente per generare tutte le istanze, bisogna variare δ

I due modelli tendono a somigliarsi per $p = \delta$

Istanze casuali: grafi

Grafi casuali con un dato numero di vertici n

1. modello di Gilbert $G(n,p)$, ovvero a probabilità uniforme p:

$$Pr[(i,j) \in E] = p \qquad (i \in V, j \in V \setminus \{i\})$$

   I grafi con un dato numero di lati m hanno tutti la stessa probabilità $p^m (1-p)^{n(n-1)/2-m}$ (diversa per ogni m)

   Se $p = 0.5$, coincide col modello a equiprobabilità di tutti i grafi

2. modello di Erdős-Rényi $G(n,m)$: dato il numero di lati m, si estraggono m coppie non ordinate di vertici casualmente in modo uniforme e si crea un lato per ciascuna

   Ovviamente per generare tutte le istanze, bisogna variare m

I due modelli tendono a somigliarsi per $p = \dfrac{2\,m}{n(n-1)}$

Istanze casuali: funzioni logiche

CNF casuali con un dato numero di variabili n

Nel loro studio, spesso si introduce un secondo parametro dimensionale:

- k è il numero di letterali per ogni formula logica

Le distribuzioni casuali più usate sono:

1. fixed-probability ensemble: si scorrono tutte le $\binom{n}{k}2^k$ possibili formule di k letterali distinti e coerenti e per ciascuna si decide con probabilità p se aggiungerla
2. fixed-size ensemble: dato il numero m di formule desiderato, per ogni formula si sommano k letterali distinti e coerenti, estratti casualmente in modo uniforme

I due modelli non differiscono molto, se $p = \dfrac{m}{\binom{n}{k}2^k}$

Transizioni di fase

Valori diversi dei parametri della distribuzione di probabilità corrispondono a regioni diverse dello spazio delle istanze

Per i grafi, ad esempio

- $m = 0$ e $p = 0$ corrispondono ai grafi vuoti
- $m = n(n-1)/2$ e $p = 1$ corrispondono ai grafi completi
- valori intermedi corrispondono a grafi di densità intermedia (rigorosamente per i valori di m, probabilisticamente per p)

In molti problemi si osserva che le prestazioni degli algoritmi sono completamente diverse in regioni differenti per quanto riguarda

- il tempo di calcolo (per gli algoritmi sia esatti, sia euristici)
- la qualità della soluzione (per gli algoritmi euristici)

La variazione di comportamento avviene bruscamente in regioni ristrette nello spazio dei parametri, come nelle transizioni di fase dei sistemi fisici

Ci sono stime dei valori critici basate sulla teoria e stime empiriche

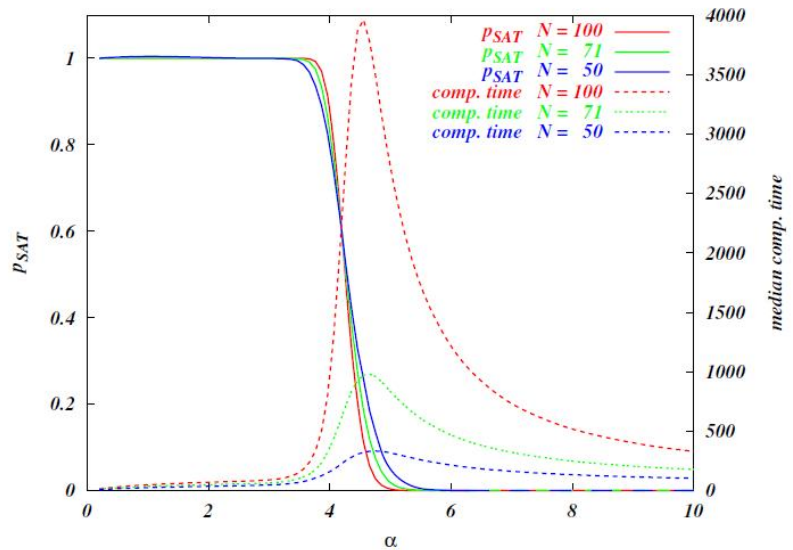
La transizione di fase per 3-*SAT*

Per semplicità, discutiamo un problema di decisione:

Data una *CNF* su n variabili, composta da formule logiche di 3 letterali, esiste un assegnamento di verità che la soddisfi?

Al crescere del rapporto formule/variabili, $\alpha = m/n$

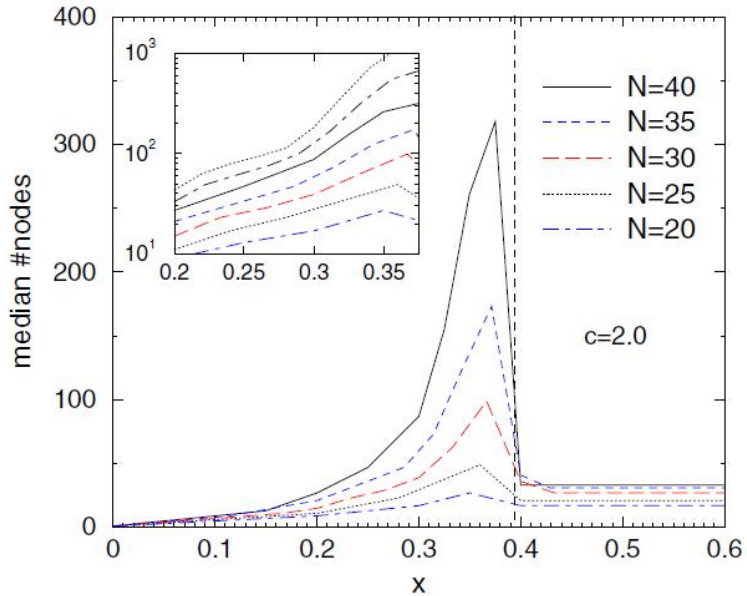
- le istanze soddisfacibili calano da quasi tutte (molte variabili e poche formule) a quasi nessuna (poche variabili e molte formule)
- il tempo di calcolo parte basso, esplode e poi torna basso
  (*si usa un noto algoritmo esatto per il* 3-*SAT*)



Per $n \to +\infty$, la transizione si fa più rapida e tende ad $\alpha_c \approx 4.26$

La transizione di fase per il *VCP*

Fenomeni simili si verificano per i problemi di ottimizzazione



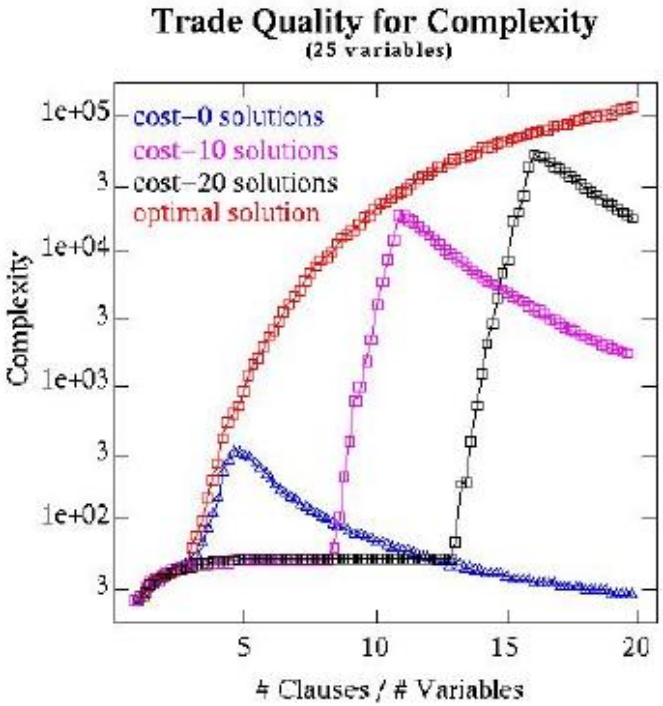
Al crescere di $\frac{|x|}{|V|}$ (qui parametrizziamo sulla cardinalità della soluzione)

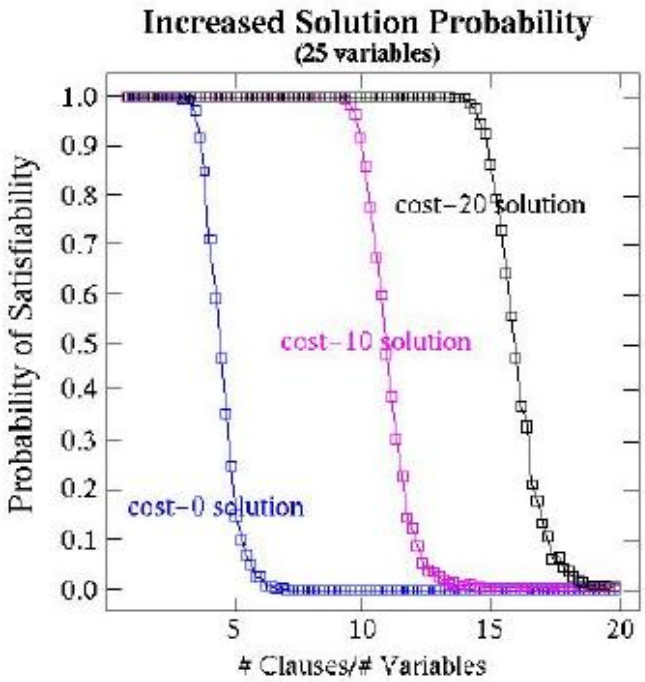
- il tempo di calcolo cresce esponenzialmente, poi crolla
- la transizione si fa più rapida e tende a un valore critico

Anche qui, si usa un noto algoritmo esatto

La transizione di fase per *Max-SAT*

Non sempre la transizione di fase separa due zone facili con una difficile
è anche frequente passare da una zona facile e una difficile





In questi casi, spesso la distinzione chiave è tra trovare una soluzione molto buona (magari velocemente) e dimostrare che sia ottima

E gli algoritmi euristici?

Abbiamo visto molti esempi riferiti ad algoritmi esatti

Gli algoritmi euristici molto spesso hanno complessità

- strettamente polinomiale (e con esponenti bassi)
- abbastanza robusta rispetto a parametri secondari delle istanze

Se usano passi casuali o memoria

- in genere vengono eseguiti ripetutamente sulla stessa istanza, dato che possono produrre soluzioni diverse ogni volta
- potrebbero proseguire indefinitamente (in realtà sino a cadere in un ciclo infinito per la ripetizione di un numero casuale o di una configurazione della memoria)
- il loro tempo di calcolo globale non è definito intrinsecamente (*quello della singola ripetizione sì*)
- tipicamente la loro terminazione viene imposta dall'utente
  1. fissando un tempo o un numero di ripetizioni desiderato
  2. fissando condizioni legate alla qualità dei risultati (per es., un valore target o un tempo massimo dopo l'ultimo miglioramento)

E gli algoritmi euristici?

E allora perché ne abbiamo parlato?

1. per orientare la scelta dell'algoritmo: un algoritmo esatto inefficiente nel caso pessimo potrebbe essere efficiente nel caso specifico
2. per far interagire utilmente algoritmi esatti ed euristici: un algoritmo euristico può fornire la soluzione che dimostra di poter applicare un algoritmo esatto nel caso specifico
3. perché la *kernelization* è di per sé utile, dato che produce istanze su cui un algoritmo euristico diventa più efficiente ed efficace
4. le istanze che richiedono molto tempo agli algoritmi esatti talvolta si rivelano difficili anche per gli algoritmi euristici,
   - non nel senso che richiedono molto tempo
   - ma nel senso che forniscono risultati peggiori

(*ma ci sono anche istanze facili per un algoritmo e difficili per un altro*)