Teoria della Complessità
Concetti fondamentali

- la teoria della complessità computazionale tenta di rispondere a domande del tipo

  quanto è efficiente un algoritmo ?

  quanto è intrinsecamente difficile un problema ? Cioè, quanto la difficoltà di un certo problema è una caratteristica generale e non è associata a particolari casi del problema stesso ?

Problema : domanda (circa le proprietà di una struttura matematica) la cui risposta dipende dai valori assunti da un insieme di **parametri** e dai valori assunti da un insieme di **variabili**

Istanza di un problema

Un particolare caso del problema per cui sono stati **specificati** tutti i valori assunti dai **parametri**

Un problema può essere descritto come l'insieme delle sue infinite istanze.

Esempio 1

Problema: Qual è il minimo fra n interi $c_1, c_2, \ldots, c_n$

Istanza: $n = 5$, $c_1 = 2$, $c_2 = 10$, $c_3 = 7$, $c_4 = 4$, $c_5 = 9$

Esempio 2

Problema: Dato un grafo $G=(V,E)$, esiste un ciclo hamiltoniano di lunghezza $<K$?

Istanza : un particolare grafo $G=(V,E)$, con particolari nodi, archi ei pesi degli archi, un valore $K>0$

Esempio 3

Problema: Dire se un intero positivo k è primo

Istanza : $k = 23456789237689$

Dimensione di un'istanza

La dimensione di un'istanza è il numero di bit necessari a codificarla.

Quindi la ***dimensione*** **dipende** dalla ***codifica*** scelta

Esempio 1:
Codifica decimale (*ragionevole*)

1 cifra (0-9) $\Rightarrow$ 10 simboli
10 (2 cifre) $\Rightarrow$ 100 simboli
100 (3 cifre) $\Rightarrow$ 1000 simboli
1000 (4 cifre) $\Rightarrow$ 10000 simboli

$\Rightarrow$ mentre il numero di simboli cresce di fattori 10
il numero di cifre cresce come il logaritmo

Dimensione di un'istanza

La dimensione di un'istanza è il numero di bit necessari a codificarla.

Quindi la ***dimensione*** **dipende** dalla ***codifica*** scelta

Esempio 2:

Codifica unaria (*non ragionevole*)

1 cifra (1) $\Rightarrow$ 1 simbolo
10 decimale (10 cifre) $\Rightarrow$ 10 simboli
100 decimale (100 cifre) $\Rightarrow$ 100 simboli
1000 decimale (1000 cifre) $\Rightarrow$ 1000 simboli

$\Rightarrow$ mentre il numero di simboli cresce di fattori 10
il numero di cifre cresce linearmente

Codifica Ragionevole

1. utilizza almeno 2 simboli

Esempio 1

Problema: Qual è il minimo fra n interi $c_1, c_2, \ldots, c_n$

Istanza: $n = 5$, $c_1 = 2$, $c_2 = 10$, $c_3 = 7$, $c_4 = 4$, $c_5 = 9$

Codifica: binaria

Dimensione: $\lceil \log_2 n \rceil + \sum_{j=1}^{n} \lceil \log_2 c_j \rceil$

Esempio 3

Problema: Dire se un intero positivo k è primo

Istanza : k = 23456789237689

Codifica: binaria

Dimensione: $\lceil \log_2 k \rceil$

Codifica Ragionevole

2. non introduce dati irrilevanti né richiede una generazione esponenziale di dati

Esempio (irragionevole): Generazione esponenziale di dati

Problema (di decisione difficile):
dato un grafo determinare se esiste un ciclo hamiltoniano di lunghezza <*K*.

Problema (di decisione semplice):
dati *n* numeri trovare se esiste un numero <*K* (sorting)

Con una codifica **non** ragionevole (codificare il grafo e tutti i suoi cicli) si potrebbe trasformare il problema difficile del ciclo hamiltoniano nel problema semplice del sorting.

La codifica non è ragionevole perché in essa è contenuta la generazione delle soluzioni del problema.

Assunzione 1: Nel seguito assumeremo sempre l'utilizzo di una codifica ragionevole delle istanze di un problema

Inoltre in generale sarà sufficiente individuare l'**ordine di grandezza** della dimensione delle istanze

Esempio

Problema: Qual è il minimo fra n interi $c_1, c_2, \ldots, c_n$

Dimensione: $O(n \log_2 L)$ dove $L = \max\{c_1, c_2, ..., c_n\}$

Esempio

Problema: Qual è il minimo fra n interi $c_1, c_2, \ldots, c_n$

con $|c_j| < 2^{31}$ per ogni $j=1,\ldots,n$

Dimensione: $O(n)$

Dato un algoritmo per risolvere un problema vogliamo determinarne la complessità computazionale come funzione della dimensione dell'istanza che si vuole risolvere

Complessità computazionale

Complessità in Tempo : **tempo di calcolo** richiesto per risolvere un'istanza del problema

Complessità in Spazio : quantità di memoria richiesta per trovare la soluzione

Assunzione 2: Nel seguito faremo principalmente riferimento alla Complessità in Tempo.

Per definire il **tempo di calcolo** è necessario far riferimento ad un qualche **modello di calcolo**

Assunzione 3: Nel seguito assumeremo **un modello di calcolo deterministico e sequenziale**, ad esempio: macchina di Turing, oppure Random Access Machine (RAM).

Per definire il **tempo di calcolo** è inoltre necessario definire il **criterio di costo** utilizzato

criterio di costo uniforme: ciascuna istruzione elementare di un algoritmo richiede una unità di tempo indipendentemente dalla dimensione degli operandi

criterio di costo logaritmico: il tempo richiesto da ciascuna istruzione elementare di un algoritmo dipendente dal numero di bit necessari a rappresentare gli operandi

Con il **criterio di costo uniforme**

Tempo di calcolo: numero delle **istruzioni elementari (**addizioni, moltiplicazioni, confronti**)** necessarie all'algoritmo per risolvere un'istanza di una dimensione data**.**

Con il **criterio di costo logaritmico**

Tempo di calcolo: somma dei **costi logaritimici** delle **istruzioni elementari (**addizioni, moltiplicazioni, confronti**)** necessarie all'algoritmo per risolvere un'istanza di una dimensione data**.**

Esempio

Problema: Dato un intero *n*, quanto vale $z=3^{2^n}$

Algoritmo:

```
read n
y := 3
while n > 0 do
    y := y*y
    n := n - 1
write y
```

Tempo di calcolo:

criterio di costo uniforme **:** $T(n) = \Theta(n)$

criterio di costo logaritmico **:** dopo la *k*-esima iterazione

la dimensione dell'operando *y* vale $\log_2 3^{2^k}$,

l'operando *n* vale $n-k$

quindi $$T(n) = \Theta\left(\sum_{k=0}^{n-1}\left(2^k \log_2 3 + n - k\right)\right) = \Theta\left(2^n\right)$$

Assunzione 4: Nel seguito assumeremo il criterio di costo uniforme

Il tempo di calcolo dipende anche dalla singola istanza considerata:

Assunzione 5: Nel seguito assumeremo l'analisi di complessità nel caso peggiore: worst case analysis

Riassunto assunzioni:

1. l’utilizzo di una codifica ragionevole delle istanze
2. il riferimento alla Complessità in Tempo
3. un modello di calcolo deterministico e sequenziale
4. il criterio di costo uniforme
5. l’analisi di complessità nel caso peggiore

Classificazione degli algoritmi

Definizione

Un algoritmo è di tipo **polinomiale** se ha una funzione di complessità nel tempo $T(n)$ che è $O(n^p)$ per un certo p fissato.

Definizione

Un algoritmo è di tipo **esponenziale** se ha una funzione di complessità nel tempo non limitata superiormente da un polinomio.

Definizione

Un algoritmo è **efficiente** se è **polinomiale**

La distinzione fra algoritmi **efficienti** e **non efficienti** è fondata

Esempio: tre algoritmi su un calcolatore che esegue una operazione in 10^{-6}s

O(f(k)) \ k	10	30	60
k^2	10^{-4}s	$9 \cdot 10^{-4}$s	$3.6 \cdot 10^{-3}$s
k^5	0.1s	24.3s	13m
2^k	10^{-3}s	17.9m	366c

polinomiale

esponenziale

Legenda: s=secondi, m=minuti, c=secoli (!)

La complessità computazionale non dipende dalla velocità del calcolatore utilizzato.

Esempio: un calcolatore 1000 volte più veloce (10^{-9}s) quante operazioni potrebbe eseguire in più per i tre algoritmi?

	10^{-6}	10^{-9}
k^2	n	$3162 \cdot n$
k^5	n	$3.98 \cdot n$
2^k	n	$10 + n$

Classificazione dei problemi

Problemi di decisione

Risposta SI/NO

(es., $\exists$ un ciclo hamiltoniano di lunghezza <K?)

Problemi di ricerca

Trovare una soluzione (una prova della risposta “SI”)

(es., trovare un ciclo hamiltoniano di lunghezza <K)

Problemi di enumerazione

Trovare tutte le soluzioni

(es., trovare tutti i cicli hamiltoniani di lunghezza <K)

Definizione

Un problema si dice **polinomiale** (o facile) se esiste un algoritmo **polinomiale** (efficiente) che lo risolve

Stabilire la difficoltà di un problema: misurare l'efficienza del (migliore) algoritmo risolutore

Classe *P*

Appartengono alla **classe *P*** tutti i problemi di **decisione** che possono essere risolti da algoritmi di tipo **polinomiale** (Polynomial time algorithm).

Esempio:

Sono algoritmi polinomiali gli algoritmi per determinare il minimo albero ricoprente in un grafo:

Kruskal è O($n$log$n$), Prim è O(n^2)

Il problema (di decisione) dello shortest spanning tree: “dato un grafo stabilire se esiste un albero ricoprente di lunghezza <K” è in ***P***

Più delicata è la questione relativa ai problemi *probabilmente difficili*, per i quali non è noto, ad oggi, alcun algoritmo polinomiale, ma nulla esclude che esso esista

Esempio: la programmazione lineare (Dati $\boldsymbol{A}$,$\boldsymbol{b}$,$\boldsymbol{c}$ e k, $\exists\ \boldsymbol{x} \geq \mathbf{0}$, t.c. $\boldsymbol{Ax} = \boldsymbol{b}$, $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} \leq k$?) è stata considerata difficile fino al 1979, quando L.J. Khachiyan presentò un algoritmo di risoluzione polinomiale (*metodo dell'elissoide*)

La **distinzione** fra **problemi facili** e **difficili** è legata allo **stato dell'arte**, anche se si può **congetturare** che alcuni problemi siano intrinsecamente più difficili degli altri

La teoria della complessità computazionale cerca di risolvere queste ambiguità definendo una gerarchia di difficoltà intrinseca dei problemi

Il primo passo è quello di introdurre un modello di calcolo (più potente) adottando il quale risultino facili i problemi per i quali non è noto, ad oggi, alcun algoritmo polinomiale

Modello di calcolo non deterministico (**e sequenziale**), ad esempio: macchina di Turing **non** deterministica.

Una macchina di Turing **non** deterministica è una macchina di Turing nella quale si sostituisce

alla **funzione** di transizione che, noto lo stato della macchina e il contenuto della cella corrente nel nastro di lavoro, **determina univocamente** l'azione da intraprendere

una **relazione** di transizione che, noto lo stato della macchina e il contenuto della cella corrente nel nastro di lavoro, **sceglie** (in modo non deterministico) l'azione da intraprendere in un insieme di azioni alternative

Il secondo passo è quello di classificare i problemi che possono essere efficientemente risolti mediante il nuovo modello di calcolo

Classe *NP* (Nondetrerministic Polynomial)

Appartengono alla **classe *NP*** tutti i problemi di **decisione** che possono essere risolti da algoritmi di tipo **polinomiale** mediante una macchina di Turing **non** deterministica

Dobbiamo ora elaborare nuove tecniche di analisi degli algoritmi utilizzando il modello di calcolo non-deteministico, buttando al vento tutta l'esperienza accumulata con il modello deterministico???

NO ! (per fortuna)

Definizione

Si definisce **certificato polinomiale** una informazione ausiliaria che può essere utilizzata per **verificare**, in **tempo polinomiale** nella dimensione dell'istanza, la **correttezza** della **risposta** per una data istanza

Non è importante ***come*** ottenere il certificato, ma la sua ***esistenza***

Classe *NP* (Definizione equivalente)

Appartengono alla classe ***NP*** tutti i problemi di decisione per i quali esiste un **certificato polinomiale** in corrispondenza di ogni loro istanza "SI", che ammette cioè una risposta affermativa

Esempio:

Problema:

Dato il grafo G, esiste un circuito hamiltoniano di lunghezza $< K$?

Risposta:

Si

Certificato polinomiale :

Una sequenza di nodi del grafo

Verifica polinomiale :

Seguire la sequenza dei nodi verificando che corrisponde ad un circuito hamiltoniano di lunghezza $< K$

Commento: esistono problemi non in NP (tipicamente quelli enumerativi) e problemi per cui non esiste modo di verificare la correttezza di una soluzione data

Un algoritmo per la classe ***NP***:

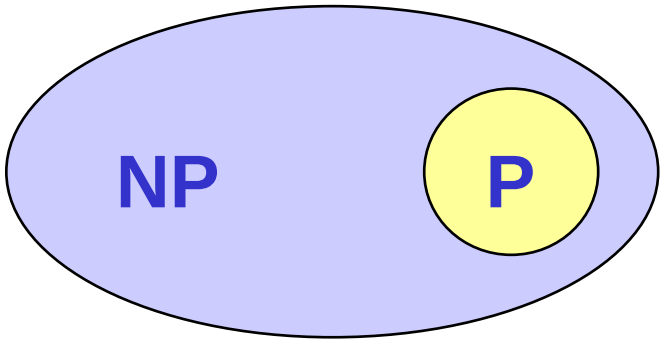
Guessing stage) Un oracolo "indovina" la soluzione (non deterministico)

Checking stage) Verifica in tempo polinomiale della soluzione

(ovviamente nella realtà tali algoritmi non esistono)

Teorema $P \subseteq NP$

Congettura $P \neq NP$



Esempi

- Shortest spanning tree: è in ***P*** e quindi in ***NP***
- Programmazione Lineare: è in ***NP*** e nel 1979 è stato dimostrato che è anche in ***P***
- Ciclo hamiltoniano di lunghezza < *K*: è in ***NP*** ma non è ancora stato dimostrato che sia in ***P***

Ma che cosa accade se i dati di input di un problema di decisione sono tali che la risposta è NO

In questo caso vedremo che c'è una sostanziale **asimmetria** fra i problemi in ***P*** e quelli in ***NP***

Per meglio comprendere tale asimmetria è necessario passare attraverso il concetto di problema complementare

Problema complementare (co-problema)

In un problema decisionale π le cui soluzioni x in F devono soddisfare una data proprietà P, i.e., $x \in F$ se e solo se $P(x)$ =VERO, si deve rispondere alla seguente domanda

$$\exists\, x \text{ t.c. } P(x) = \text{VERO ?}$$

nel problema complementare co-π si deve rispondere alla domanda "negata", ovvero se è falsa l'affermazione precedente

$$\neg\, \exists\, x \text{ t.c. } P(x) = \text{VERO ?} \quad \Rightarrow \forall\, x \quad P(x) = \text{FALSO ?}$$

Esempio:

Il problema complementare del ciclo hamiltoniano di lunghezza < K:

"Dato il grafo G, è vero che non esistono cicli hamiltoniano di lunghezza < K ?"

Classe co-*NP* (*NP* complementare)

Appartengono alla classe **co-*NP*** tutti i problemi di decisione per i quali esiste un **certificato polinomiale** in corrispondenza di ogni loro istanza "NO"

Esempio:

Il certificato di ammissibilità per il problema complementare del ciclo hamiltoniano:

la lista di tutti i possibili cicli hamiltoniani

Per un grafo completo con m nodi è pari a $m!$ $\Rightarrow$ esponenziale $\Rightarrow$ **non** verificabile in tempo polinomiale

"Probabilmente"

il problema del ciclo hamiltoniano $\notin$ **co-*NP***

Osservazione:

Dato un problema decisionale π fornire un certificato polinomiale per il corrispondente problema co-π è in generale più difficile che fornire un certificato polinomiale per π

(**co-π non sembra $\in$ *NP***) $\equiv$ (π **non sembra $\in$ co-*NP***)

Osservazione:

Esistono problemi in cui è facile dimostrare la correttezza sia del problema che del co-problema. Questa situazione si verifica quando $\pi \in \boldsymbol{P}$. In tal caso infatti è possibile **determinare** una soluzione per il problema decisionale, o negarne l'esistenza, in tempo polinomiale (il certificato è vuoto)

Esempio:

π : esiste un percorso tra Milano e Brescia minore di 100 Km?

***co*-π : tutti i percorsi tra Milano e Brescia sono *non* minori di 100 Km?**

Per rispondere, calcoliamo in tempo polinomiale la distanza minima, **d**, tra Milano e Brescia, se **d** < `100`, la risposta a π è SI e a **co**-π è NO, altrimenti, se **d** ≥ `100`, la risposta a π è NO e a **co**-π è SI

Si può determinare la correttezza della risposta a **co**-π in tempo polinomiale, in questo caso **co**-π ∈ ***NP***.

I problemi complementari dei problemi in ***P*** sono in ***P***

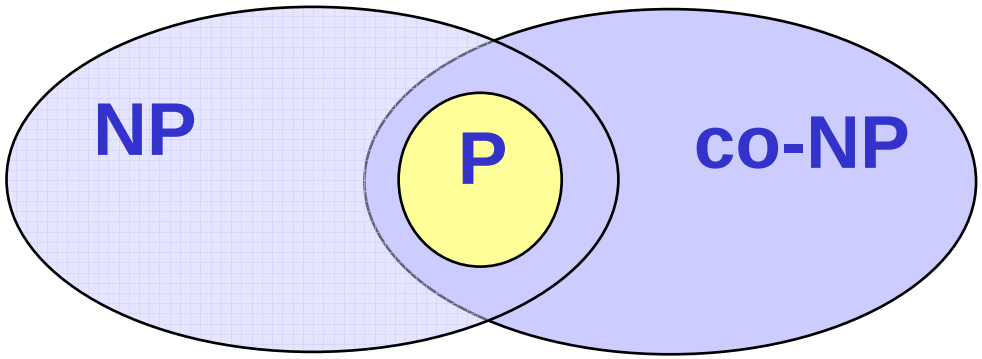
Non tutti i problemi complementari dei problemi in ***NP*** sembrano essere in ***NP***

Teorema P = co-P

Teorema $P \subseteq NP \cap co\text{-}NP$

Congettura $NP \neq co\text{-}NP$

Congettura $P = NP \cap co\text{-}NP$



Problemi di decisione e di ottimizzazione

- Le definizioni della teoria della complessità fanno riferimento a problemi di decisione.
- Quanto finora detto si può estendere ai problemi di ottimizzazione ?

In particolare un algoritmo che risolve un problema di decisione mediante un certificato polinomiale può essere utilizzato all'interno di un algoritmo di tipo polinomiale per risolvere il corrispondente problema di ottimizzazione ?

Esempio:

Problema di decisione (D(k))

Dato il grafo G, esiste un circuito hamiltoniano di lunghezza <K?

Problema di ottimizzazione (TSP)

Determinare il ciclo hamiltoniano di G di lunghezza minima.

Sia S l'algoritmo per risolvere D(k) mediante un certificato polinomiale. L'algoritmo per risolvere TSP può essere costituito dalla seguente procedura dicotomica: ?

1. Siano A e B la minima e massima lunghezza degli archi di G ed n il numero di nodi di G.
   Allora $n \times A \leq$ lunghezza circuto hamiltoniano $\leq n \times B$
   Porre $k = n \times (B - A) / 2$
2. Usare S per risolvere D(k).
3. **if** la risposta di S è SI, porre k=k/2 **else** k=k(3/2) ed andare a 2.

Poichè i coefficienti sono interi, al più ci sarebbero $\log(n \times (B - A)) + 1$ iterazioni...

L'idea **non** funziona! Infatti al passo 3 l'algoritmo in caso di risposta NO (ramo **else**) non risponde necessariamente in tempo polinomiale (mediante un certificato polinomiale).

Questo fatto è dovuto alla sostanziale **asimmetria** fra i problemi in ***NP*** e quelli in **co-*NP***:

la tecnica della ricerca dicotomica rende equivalenti i problemi di decisione e quelli di ottimizzazione solo nel caso dei problemi nella classe ***P*** (per i quali vale **P = co-P**), ma non può essere usata per tutti i problemi della classe ***NP***

Torniamo ora alla delicata questione relativa ai problemi *probabilmente difficili*, per i quali non è noto, ad oggi, alcun algoritmo polinomiale, ma nulla esclude che esso esista

Come possiamo usare le nozioni fin qui introdotte per supportare la congettura che alcuni di questi problemi siano intrinsecamente più difficili degli altri ? E in che senso più difficili ?

Definizione

Si definiscono come problemi più difficili di una classe (Problemi completi) quei problemi appartenenti alla classe che risultano almeno difficili quanto ogni altro

Se si riuscisse a risolvere efficientemente uno dei problemi più difficili allora si potrebbero risolvere efficientemente anche tutti gli altri problemi della classe

Ma come possiamo dire che un problema appartiene all'insieme dei problemi più difficili della sua classe ?

Mediante il concetto di

riduzione polinomiale fra problemi

Trasformazione polinomiale (Cook)

Il problema π si **trasforma** in tempo polinomiale nel problema π' se esiste un algoritmo che, data un'istanza I di π, produce in tempo polinomiale un'istanza I' di π' in modo tale che la risposta è "SI" per I se e solo se la risposta è "SI" anche per I'

Riduzione polinomiale

Il problema π si **riduce** in tempo polinomiale al problema π' ($\pi \propto \pi$') se esiste un algoritmo per risolvere π che

1. utilizza come procedura un qualunque algoritmo A per π'
2. risulta polinomiale nell'ipotesi che A richieda tempo O(1)

N.B. Le trasformazioni polinomiali sono **transitive**

Osservazioni

– π può essere ridotto a π’ se la soluzione del problema π può essere ottenuta risolvendo un numero polinomiale di problemi π’

– Una trasformazione o una riduzione polinomiale permette di dire che il problema π’ è almeno altrettanto difficile di π, cioè

$$\pi \text{ difficile}, \pi \propto \pi' \Rightarrow \pi' \text{ difficile}$$

infatti se esistesse un algoritmo polinomiale A per π’ allora A potrebbe essere utilizzato come procedura per risolvere in tempo polinomiale π

Ora siamo in grado di dare la seguente fondamentale definizione

Classe *NP*-C (*NP*-Completi)

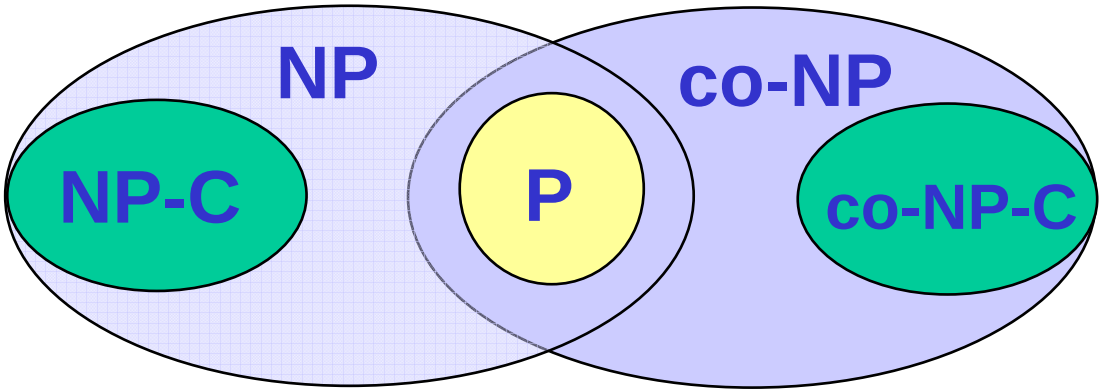
Appartengono alla classe ***NP*-*C*** tutti i problemi della classe ***NP*** che sono almeno difficili quanto ogni altro problema in ***NP***, cioè un problema $\pi' \in$ ***NP*-*C*** se

1. $\pi' \in$ ***NP***
2. $\pi \propto \pi' \; \forall \pi \in$ ***NP***

Conseguenze:

- Se fosse possibile risolvere efficientemente un $\pi \in$ ***NP-C*** allora tutti i problemi in ***NP*** sarebbero risolti efficientemente

- Per dimostrare ***P*** = ***NP*** basterebbe provare che un solo problema in ***NP-C*** è in ***P***



Il problema della soddisfacilità (SAT)

E' stato il primo problema ***NP*** di cui è stata dimostrata l'appartenenza ad ***NP-C***

SAT Problem

Data un'espressione booleana F in forma congiuntiva normale (CNF) con n variabili, $x_1, \ldots, x_n$, e loro complementi, dire se F è soddisfacibile

CNF : un AND di OR:

$$\bigwedge_{i=1}^{m} \left(y_{1_i} \vee \ldots \vee y_{h_i} \right) \qquad con \qquad y_{j_i} \in \left\{ x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n \right\}$$

dove le espressioni in parentesi sono clausole di cardinalità h

Esempio: Data $F = (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_4) \wedge$

$\wedge (x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$

dire se esiste un'assegnazione VERO-FALSO alle variabili per cui F = VERO.

Teorema di Cook (1971)

Il problema SAT è NP-C se h≥3.

Per dimostrare che $\pi' \in$ ***NP*** è NP-Completo, per la transitività di $\propto$, è sufficiente dimostrare che un $\pi \in$ ***NP-C*** (ad esempio, SAT) si riduce a π', cioè $\pi \propto \pi'$

Esempio

Dimostriamo che k-Clique *è NP-Completo*

Dato un grafo `G=(N,E)` esiste un sottografo completo di `G` con almeno k nodi ?

Esempio

SAT si riduce a k-Clique

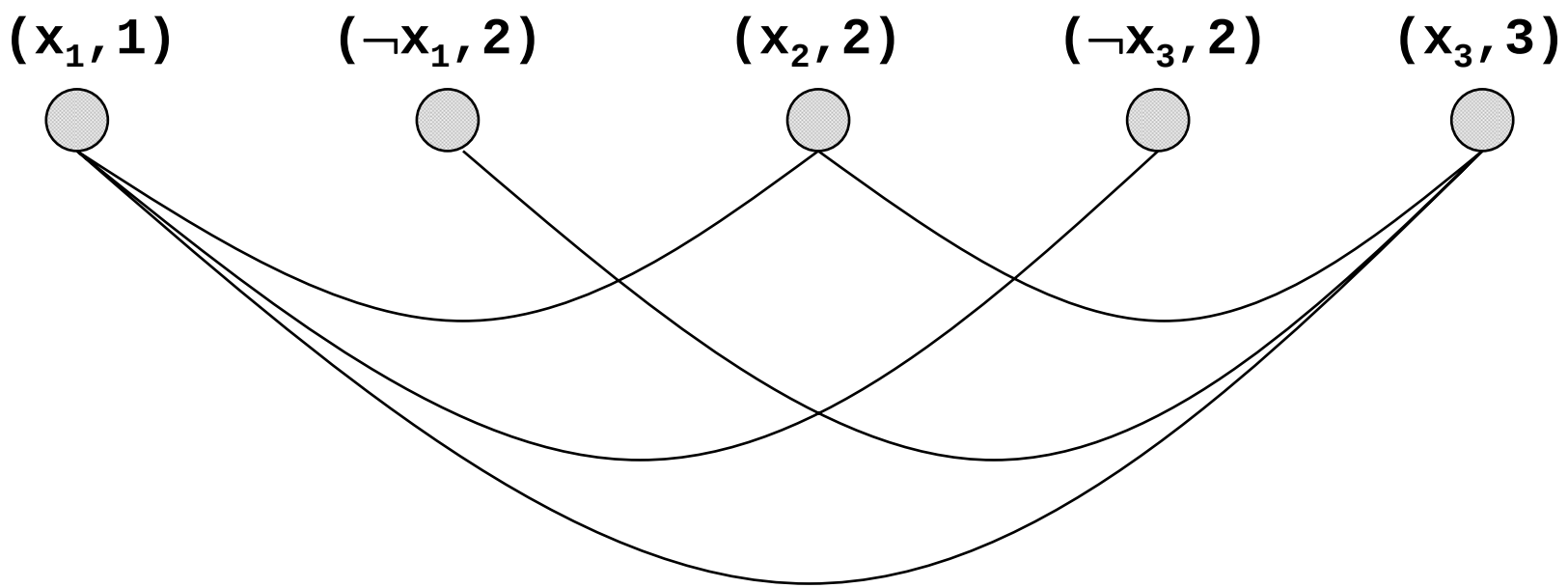
Si costruisce un grafo ausiliario G=(N,E) dove
N={(x,i) t.c. x è una variabile della i-esima clausola}
E={[(x,i),(y,j)] t.c. y $\neq \neg$x e i $\neq$j}
cioè un vertice per ogni occorrenza di una variabile nell'istanza di *SAT* e un lato per ogni coppia di occorrenze che appartengano a clausole diverse e che non corrispondano a letterali opposti

Una clique di k vertici corrisponde a k occorrenze di variabili fra loro coerenti (senza cioè che una variabile compaia sia affermata che negata) distribuite in k clausole distinte: se esiste, si soddisfano le k clausole (per k = m, l'intera espressione)

Se non esiste, nessun assegnamento è soddisfacente, dato che un assegnamento soddisfacente determina una clique

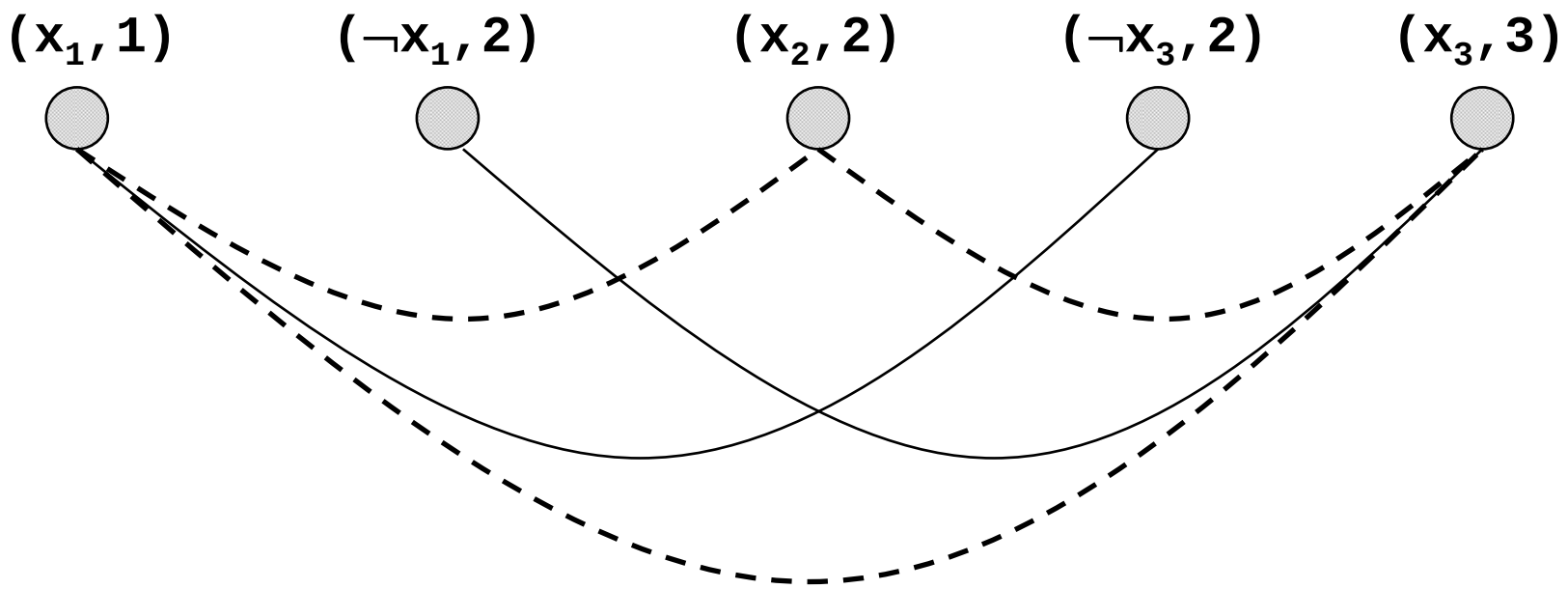
Esempio

$$F = (x_1) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3)$$



Esempio

$$F = (x_1) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3)$$



3-SAT Problem (3-SAT)

Data un'espressione booleana F in forma congiuntiva normale (CNF) con n variabili, $x_1, \ldots, x_n$, e loro complementi, e al più tre variabili per clausola, dire se F è soddisfacibile

Set partitioning Problem (SP)

Dato un insieme finito S e una famiglia F di sottoinsiemi di S, dire se esiste un sottoinsieme di F formato da membri a due a due disgiunti la cui unione da S

Esempio:

Il problema dello zaino 0-1, KP, è in ***NP-C***, infatti si dimostra

che SAT $\propto$ 3–SAT, che 3–SAT $\propto$ Clique, che Clique $\propto$ SP e infine che SP $\propto$ KP,

quindi poiché $\forall \pi \in$ ***NP*** vale che $\pi \propto$ SAT allora segue che $\forall \pi \in$ ***NP*** vale che $\pi \propto$ KP

Estensione ai problemi di ottimizzazione.

Classe *NP*-hard (*NP*-difficili)

Appartengono alla classe ***NP*-hard** tutti i problemi che sono almeno difficili quanto ogni problema in ***NP***, cioè un problema

$\pi' \in$ ***NP*-hard** se $\pi \propto \pi' \; \forall \pi \in$ ***NP***

Sono ***NP*-hard** problemi almeno difficili quanto ogni problema ***NP*-*C***. Sono ***NP*-hard** le versioni di “ottimizzazione” dei problemi ***NP*-*C***

Problemi NP-C (NP-hard) in senso debole/forte

Algoritmo pseudopolinomiale:
algoritmo che risolve un problema NP-C (o NP-hard) in un tempo che sia polinomiale nella lunghezza dell'input quando i coefficienti sono codificati in maniera "unaria"

Problema NP-C (NP-hard) in senso debole:
problema NP-C (NP-hard) che può essere risolto da un algoritmo pseudopolinomiale
Esempio **Il problema dello zaino 0-1**

Problema NP-C (NP-hard) in senso forte:
problema NP-C (NP-hard) che non è NP-C (NP-hard) in senso debole
Esempio **SAT**

Teorema

Se un problema NP-C in senso forte viene risolto da un algoritmo pseudo polinomiale allora P = NP

Teorema

Se un problema complementare co-π di un qualunque problem $\pi \in$NP-C è in NP , cioè co-$\pi \in$NP allora NP = co-NP

Se non conosciamo un algoritmo polinomiale che risolve un certo problema π, e non lo abbiamo classificato ***NP***-completo ma sappiamo che $\pi \in$ ***NP*** $\cap$ ***co-NP***, siamo più portati a credere che ciò dipenda da una nostra incompleta conoscenza algoritmica piuttosto che dal fatto che il problema è intrinsecamente difficile

Esempio

Problema: Dire se un intero positivo *k* è primo

[Solovay, Strassen 77; Rabin 80] certificato di compostezza $\pi \in$ ***co-NP***

[Atkin 86; Adelman Huang 92] certificato di primalità $\pi \in$ ***NP***

[M. Agrawal, N. Kayal, N. Saxena, 6 Agosto 2002] $\pi \in$ ***P***

Congettura P = NP $\cap$ co-NP

- Se $\pi \in$ ***NP-C*** per alcune delle sue istanze (le più grandi) non potrà essere (probabilmente) trovata la soluzione ottima in un tempo accettabile per mezzo di algoritmi esatti (enumerazione esplicita o implicita).
- In generale è possibili determinare soluzioni accettabili per $\pi \in$ ***NP-C*** per mezzo di
  - algoritmi approssimati (soluzioni sub-ottime)
  - algoritmi euristici

Esempi di problemi

Problemi P

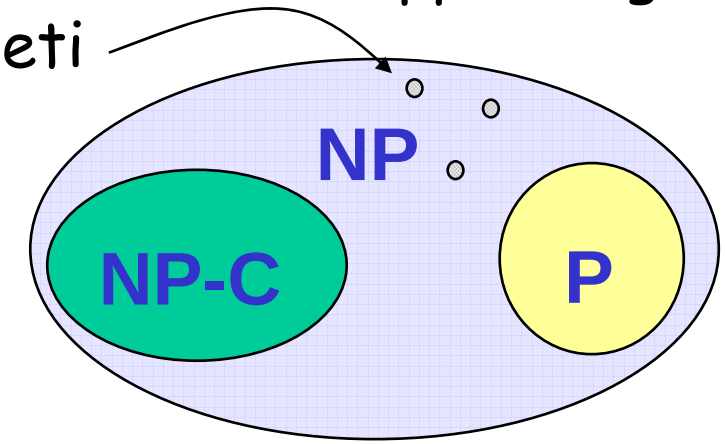
- Programmazione Lineare Continua
- Matching su grafo
- Minimum Spanning Tree
- Shortest Path (Dijkstra $O(n^2)$, Bellman-Ford $O(n^3)$)
- Sistemi di equazioni lineari

Problemi NP-C

- Travelling Salesman Problem
- SAT
- Set partitioning, packing, covering
- 0-1 Integer programming
- Knapsack

Osservazione

È' stato dimostrato che se $P \neq NP$ allora devono esistere problemi che non appartengono né a **P** né che sono NP completi



Un candidato a tale collocazione intermedia è il problema dell'isomorfismo tra grafi

Isomorfismo tra grafi

Dati due grafi $G=(V,E)$ e $G'=(V',E')$ esiste una permutazione π degli indici dei vertici di G tale che $(v_i, v_j) \in E$ se e solo se $(v_{\pi(i)}, v_{\pi(j)}) \in E'$?

Esempi di trasformazioni

Problema

Dimostrare che è NP-completo il seguente problema

Maximum path
Dato un grafo pesato $G=(V,A)$ determinare se esiste in G un cammino *semplice* da s a t (che non passa due volte per uno stesso vertice) di lunghezza $\geq K$

usando una riduzione dal seguente problema NP-completo

Hamiltonian path
Dato un grafo $G=(V,A)$ determinare se G possiede o meno un cammino hamiltoniano fra una data coppia di vertici in s e t di V

Tecnica

Immaginiamo di possedere un algoritmo che risolve in tempo polinomiale il problema Maximum path e lo usiamo per risolvere in tempo polinomiale Hamiltonian path

Soluzione

Sia $G' = (V',A')$ il grafo del quale si vuol determinare se possieda o meno un cammino hamiltoniano fra la coppia di vertici s e t.

Costruiamo un nuovo grafo pesato $G=(V,A)$ dove $V=V'$ e $A=A'$ e dove attribuiamo ad ogni arco di A un peso unitario.

Chiediamoci ora se esiste in G un cammino *semplice* da s a t di lunghezza almeno pari a $|V|-1$.

Se tale cammino esiste allora esso tocca necessariamente tutti i vertici del grafo una e una sola volta ed è quindi hamiltoniano.

Se tale cammino non esiste allora non esiste alcuna cammino hamiltoniano da s a t.