3 CENNI DI TEORIA DELLA COMPLESSITA' COMPUTAZIONALE

Scopo: Stimare l'onere computazionale per risolvere problemi di ottimizzazione e di altra natura

- Determinare l'efficienza di un specifico algoritmo *A* per risolvere un dato problema *P*
- Tentare di valutare la difficoltà intrinseca di un dato problema *P*

3.1 Complessità degli algoritmi

Scopo: stimare l'onere computazionale di algoritmi alternativi per risolvere un dato problema P al fine di selezionare quello più efficiente

Un'*istanza* I di un problema P è un caso specifico del problema

Esempio

Problema P: ordinare n numeri interi $c_1, ..., c_n$

Istanza I : $n = 3, c_1 = 2, c_2 = 7, c_3 = 5$

Principali risorse { *tempo* di calcolo; *spazio* di memoria

Facendo riferimento ad un modello di calcolo astratto come la macchina di Turing, si considera il numero di *operazioni elementari* (aritmetiche, confronti,...) necessarie per risolvere una data istanza I

Chiaramente il numero di operazioni elementari dipende dall'istanza I

Dimensione di un'istanza

La *dimensione* di un'istanza I, indicata $|I|$, è il numero di bit necessari a codificare (descrivere) I

Esempio

Istanza specificata dai valori di n e $c_1, ..., c_n$

Poiché la codifica di un intero i richiede $\lceil \log i \rceil$ bit,

$$|I| = \lceil \log n \rceil + n \cdot \lceil \log \overline{c} \rceil \quad \text{dove} \quad \overline{c} = \max_{1 \le i \le n} c_i$$

Per istanza $n = 3, c_1 = 2, c_2 = 7, c_3 = 5$

$$\Rightarrow \quad \lceil \log 3 \rceil + 3 \cdot \lceil \log 7 \rceil$$

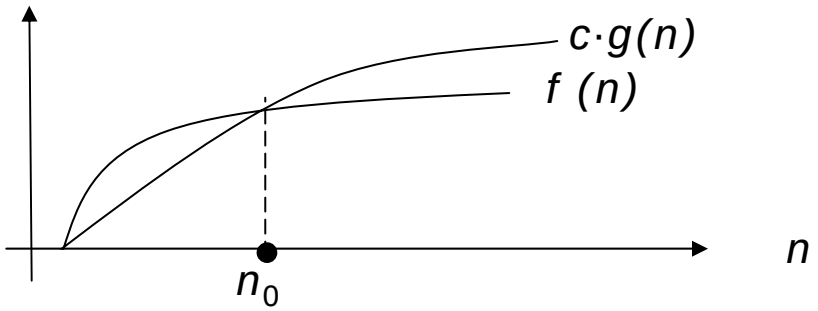
Ordine di complessità

In genere, si considera la rapidità di crescita del numero di operazioni elementari (o.e.) nel caso peggiore :

numero di o.e. per risolvere istanza $I \leq f(n) \quad \forall I$ con $|I| \leq n$

Complessità asintotica :

$f(n) = O(g(n))$ se $\exists\, c > 0$ tale che $f(n) \leq c\, g(n)$
per n sufficientemente grande



Si dice che $f(n)$ è dell'ordine di $g(n)$

Esempio

Per l'ordinamento di n interi esiste un algoritmo $O(n \log n)$

Un algoritmo è *polinomiale* se richiede, nel caso peggiore, un numero di operazioni elementari

$$f(n) = O(n^d) \quad \text{con } n = |I| \text{ e } d \text{ costante.}$$

Si distinguono algoritmi con:

$O(n^d)$	$O(2^n)$
polinomiale	esponenziale

Gli algoritmi polinomiali sono in genere considerati efficienti anche se uno in $O(n^{10})$ non lo è molto in pratica!

Esempi

- Algoritmi di Prim (alberi ottimi) e di Dijkstra (cammini minimi) complessità polinomiale: $O(n^2)$

- Versione di base dell'algoritmo di Ford – Fulkerson

Ricerca cammino aumentante: $O(m)$

Valore flusso massimo $\varphi^* \leq m\, k_{max}$ $\quad k_{max} = \max\{k_{ij}: (i,j) \in A\}$

Se k_{ij} intere, il valore φ aumenta di almeno $\delta = 1$ unità ad ogni iterazione

Partendo dal flusso di valore $\underline{x} = \underline{0}$ con $\varphi_0 = 0$, si ha quindi una complessità totale $O(m^2\, k_{max})$

N.B.: la dimensione di un'istanza è $O(m \log k_{max})$ perché la capacità di ogni arco può essere codificata su $\lceil \log k_{max} \rceil$ bit

Poiché $k_{max} = 2^{\log k_{max}}$ l'ordine di complessità totale $O(m^2 k_{max})$ non è polinomiale rispetto alla dimensione!

Modifica che lo rende polinomiale:
cercare cammino aumentante con un numero minimo di archi
(Edmonds e Karp $O(nm^2)$, Dinic $O(n^2m)$, e Malhorta, Kumar e Maheshwari $O(n^3)$)

3.2 Complessità dei problemi

Scopo: stimare la difficoltà intrinseca di un problema (che corrisponde alla complessità del "miglior algoritmo che permetterebbe di risolverlo") per adottare l'approccio risolutivo più adeguato

Un problema *P* è *polinomiale* ("*facile*") se esiste un algoritmo polinomiale che fornisce la soluzione (ottima) per ogni istanza.

Esempi: cammini minimi, flussi di valore massimo,...

Esistono problemi "difficili" ?

Per molti problemi di ottimizzazione i migliori algoritmi noti tutt'oggi richiedono un numero di operazioni elementari che cresce, nel caso peggiore, esponenzialmente con la dimensione dell'istanza

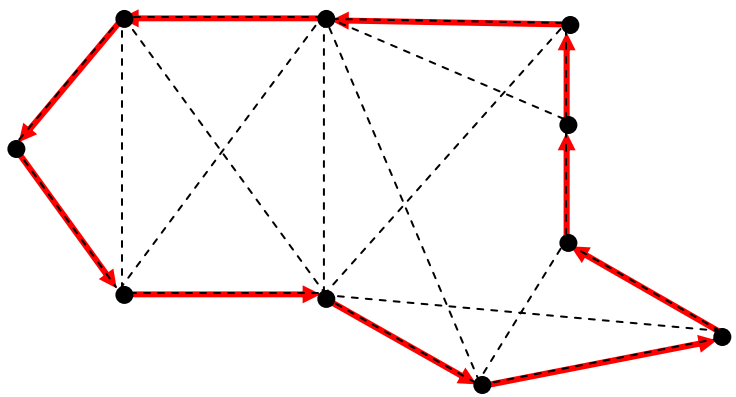
N.B.: Non dimostra che sono effettivamente "difficili"!

Problema del commesso viaggiatore

"Travelling Salesman Problem" TSP

Un commesso viaggiatore deve visitare ciascuna di n città esattamente una volta e ritornare al punto di partenza nel minor tempo possibile.

collegamenti con tempi



Problema Dato un grafo <u>orientato</u> $G = (N, A)$, con un costo $c_{ij} \in \mathbf{Z}$, $\forall\ (i, j) \in A$, determinare un <u>circuito</u> di costo minimo che visita <u>esattamente</u> <u>una volta</u> ogni nodo.

Un *circuito* C è *hamiltoniano* se passa esattamente una volta per ogni nodo.

Indicando con H l'insieme di tutti i circuiti hamiltoniani di G, il problema equivale a

$$\min_{C \in H} \sum_{(i,j) \in C} c_{ij}$$

N.B.: H contiene un numero finito di elementi

$$| H | \leq (n - 1)!$$

Applicazioni: distribuzione, sequenziamento ottimo, VLSI, …

3.3 Teoria della $\mathcal{NP}$-completezza

Non si fa direttamente riferimento ai *problemi* di ottimizzazione ma ai problemi *di riconoscimento* (risposta "si" / "no")

Ad ogni problema di ottimizzazione viene associata una versione di riconoscimento

Esempio TSP-r

Dato un grafo orientato $G = (N, A)$ con distanze c_{ij} intere e un intero L, esiste un circuito hamiltoniano di lunghezza $\leq L$?

Problemi di riconoscimento

Qualsiasi problema di ottimizzazione è almeno altrettanto difficile della sua versione di riconoscimento.

Esempio

Se si riesce a individuare un circuito hamiltoniano di lunghezza minima si può chiaramente rispondere alla domanda di TSP-r (ne esiste uno di lunghezza $\leq L$?)

Se versione di riconoscimento è difficile

$\Rightarrow$ il problema di ottimizzazione è anch'esso difficile

Classe di complessità $\mathcal{P}$

$\mathcal{P}$ indica la classe dei problemi di riconoscimento che si possono risolvere in tempo polinomiale.

Esempi: quelli associati ai problemi di alberi ottimi, di cammini minimi e di flussi massimi

Per ciascuno di essi esiste un algoritmo che permette di stabilire per ogni istanza I se la risposta è "si" o "no" in tempo polinomiale in $|I|$

Definizione formale di $\mathcal{P}$ in termini di macchina di Turing (deterministica) polinomiale

Classe di complessità $\mathcal{NP}$

Si richiede solo che per ogni istanza con risposta "si" esista una prova concisa (certificato) che permetta di verificare in tempo polinomiale che la risposta è "si".

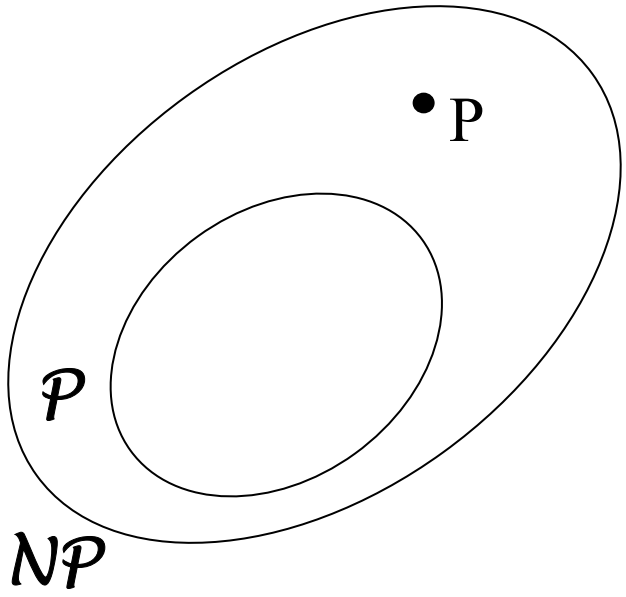
Esempio: TSP-r poiché si può verificare in tempo polinomiale se una sequenza di nodi è un circuito hamiltoniano e se lunghezza $\leq L$

$\mathcal{NP}$ indica la classe dei problemi di riconoscimento per i quali $\exists$ un polinomio $p(n)$ e un algoritmo di verifica del certificato $\mathcal{A}_{vc}$ tale che :

istanza I ha risposta "si" $\Leftrightarrow \exists$ un certificato $\chi(I)$ di dimensione polinomiale ($|\chi(I)| \leq p(|I|)$) e $\mathcal{A}_{vc}$ applicato all'input "$I,\chi(I)$" da risposta "si" in un numero di operazioni elementari $\leq p(|I|)$.

NB: Non importa quanto è difficile ottenere il certificato (può essere fornito da un "oracolo") basta che esista e sia verificabile in tempo polinomiale!

Chiaramente $\mathcal{P} \subseteq \mathcal{NP}$



Congettura $\mathcal{P} \subset \mathcal{NP}$

$\mathcal{NP}$ non sta per "Non Polinomiale" ma per "Non-deterministico Polinomiale" (fa riferimento alle macchine di Turing non-deterministiche polinomiali)

Riduzione polinomiale fra problemi

Per confrontare i problemi di riconoscimento e definire quelli più difficili si utilizza il concetto di riduzione polinomiale.

Sia P_1 e $P_2 \in \mathcal{NP}$, P_1 si *riduce in tempo polinomiale* a P_2 ($P_1 \propto P_2$) se esiste un algoritmo per risolvere P_1 che

- chiama un certo numero di volte un ipotetico algoritmo per P_2,
- e risulta polinomiale se si suppone che quello per P_2 richieda un'unica unità di tempo.

Caso più semplice: l'algoritmo per P_2 viene chiamato un'unica volta.

Esempio

P_1: Dato un grafo non orientato $G = (N, E)$ con costi e un intero L, $\exists$ un ciclo hamiltoniano di lunghezza $\leq L$?

P_2: Dato un grafo orientato $G' = (N', A')$ con costi e un intero L', $\exists$ un circuito hamiltoniano di lunghezza $\leq L'$?

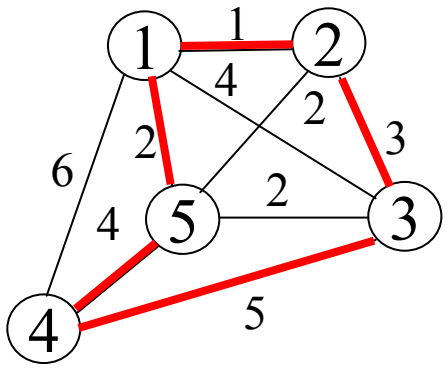
$$P_1 \propto P_2$$

Riduzione polinomiale dal caso non orientato a quello orientato :

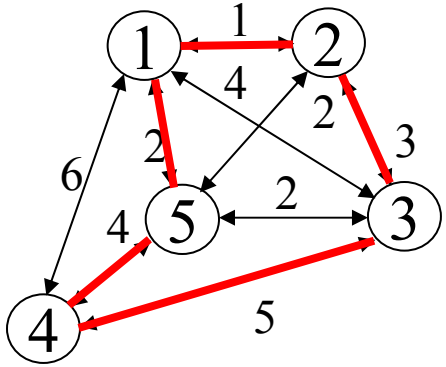
in tempo e spazio polinomiale

$\forall\ I_1 \in P_1$ è facile costruire una particolare $I_2 \in P_2$

$G=(N,E)$
$L = 15$



$G'=(N',A')$
$L' = 15$



tale che I_1 ha risposta "si" $\Leftrightarrow$ I_2 ha la risposta "si"

Conseguenza:

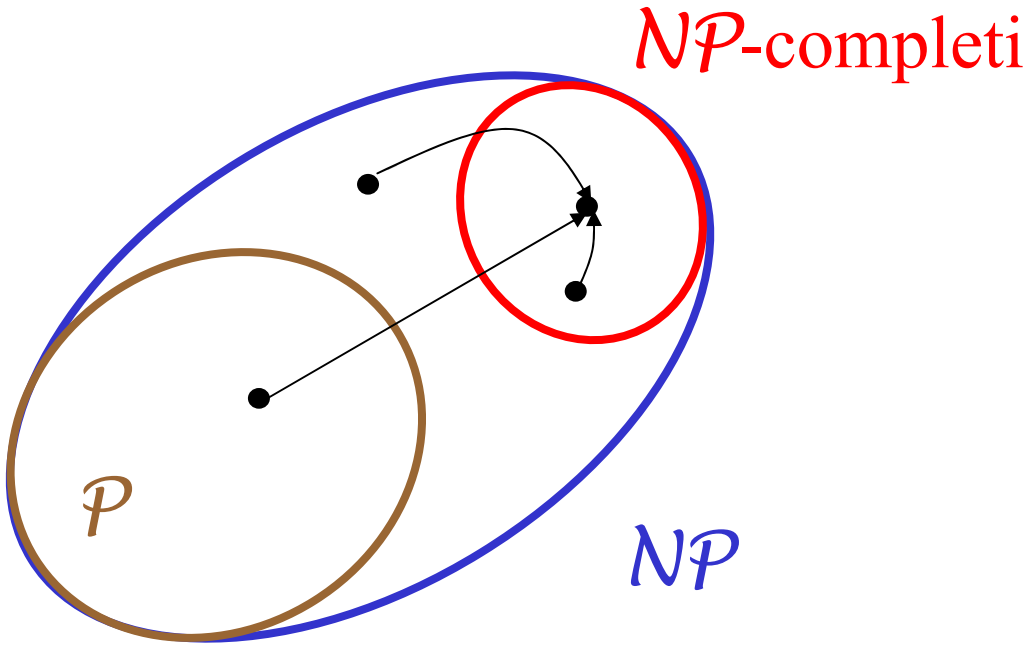
Se $P_1 \propto P_2$ e P_2 è risolubile mediante un algoritmo polinomiale, allora anche P_1 può essere risolto in tempo polinomiale.

$$P_2 \in \mathcal{P} \Rightarrow P_1 \in \mathcal{P}$$

Problemi $\mathcal{NP}$-completi

Un problema *P* è $\mathcal{NP}$-*completo* se e solo se

1) appartiene a $\mathcal{NP}$
2) ogni altro problema in $\mathcal{NP}$ è riducibile ad esso in tempo polinomiale ($P' \propto P$, $\forall$ P' $\in \mathcal{NP}$)



Proprietà:

Se un problema $\mathcal{NP}$-completo P fosse risolubile in tempo polinomiale (se $\in \mathcal{P}$), allora lo sarebbero tutti i problemi di $\mathcal{NP}$, cioè si avrebbe $\mathcal{P} = \mathcal{NP}$!!

eventualità considerata estremamente improbabile

La **$\mathcal{NP}$-completezza** è quindi un **forte indizio** di **difficoltà** intrinseca

cf. lunga lista di problemi $\mathcal{NP}$-completi per i quali non sono noti algoritmi polinomiali

Primo problema dimostrato $\mathcal{NP}$-completo (Cook 1971) :

<u>Soddisfacibilità</u> (SAT)

Date m clausole booleane $C_1,\ldots, C_m$ (disgiunzioni -- OR -- di variabili booleane y_j e loro complementi $\overline{y}_j$), esiste un assegnamento di valori "vero" o "falso" alle variabili che rende vere tutte le clausole?

Esempio

$$C_1 : (y_1 \vee y_2 \vee y_3)$$
$$C_2 : (\overline{y}_1 \vee \overline{y}_2)$$
$$C_3 : (y_2 \vee \overline{y}_3)$$

assegnamento: y_1 = vero, y_2 = falso, y_3 = falso

Come mostrare la $\mathcal{NP}$-completezza ?

Per stabilire che $P_2 \in \mathcal{NP}$ è $\mathcal{NP}$-completo "basta" mostrare che un altro problema $\mathcal{NP}$-completo P_1 si riduce polinomialmente a P_2 :

$P \propto P_1, \forall\ P \in \mathcal{NP}$, e $P_1 \propto P_2$ implica per transitività che

$$P \propto P_2, \forall\ \mathrm{P} \in \mathcal{NP}$$

Esempio

P_1: Dato G non orientato con costi e un intero L, $\exists$ un ciclo hamiltoniano di lunghezza $\leq L$?

P_2: Dato G' orientato con costi e un intero L', $\exists$ un circuito hamiltoniano di lunghezza $\leq L'$?

$$P_2 \in \mathcal{NP} \text{ e } P_1 \propto P_2 \text{ con } P_1 \ \mathcal{NP}\text{-completo}$$

Altri esempi di problemi $\mathcal{NP}$-completi

- Dato $G = (N, E)$ non orientato, ciclo hamiltoniano ?
- Dato $G = (N, A)$ orientato, due nodi assegnati s e t, e un intero L, esiste un cammino semplice (con nodi distinti) da s a t che contiene un numero di archi $\geq L$?
- Dato $G = (N, A)$ orientato con costi sugli archi, due nodi s e t, e un intero L, esiste un cammino semplice da s a t di costo $\leq L$?
- Dato un sistema lineare $A\,\mathbf{x} \geq \mathbf{b}$ con coefficienti interi, esiste una soluzione $\mathbf{x}$ a valori 0, 1 ?
- ….

Problemi $\mathcal{NP}$-difficili

Un problema è $\mathcal{NP}$-*difficile* se ogni problema in $\mathcal{NP}$ è riducibile ad esso in tempo polinomiale (anche se non appartiene ad $\mathcal{NP}$)

Esempio

TSP poiché TSP-r (esiste un circuito hamiltoniano di lunghezza $\leq L$?) è $\mathcal{NP}$-completo.

Tutti i problemi di otttimizzazione di cui la versione di riconoscimento è $\mathcal{NP}$-completa sono $\mathcal{NP}$-difficili.

Altri esempi di problemi $\mathcal{NP}$-difficili

- Dato $G = (N, A)$ orientato con costi sugli archi, due nodi assegnati s e t, determinare un cammino semplice di costo massimo tra s e t.
- Dato $G = (N, A)$ orientato con costi sugli archi, due nodi s e t, determinare un cammino semplice di costo minimo tra s e t.
- Dati matrice A $m \times n$, e vettori $\mathbf{b}$ $m \times 1$ e $\mathbf{c}$ $n \times 1$ con componenti intere, trovare un $\mathbf{x} \in \{0, 1\}^n$ che soddisfi $A\mathbf{x} \geq \mathbf{b}$ e minimizzi $\mathbf{c}^T\mathbf{x}$.
- ….

Esercizio

Qual è la dimensione di un'istanza del problema dell'albero di supporto di costo minimo ?

Esercizio

Verificare che la programmazione lineare intera è un problema $\mathcal{NP}$-difficile.

Programmazione Lineare Intera (PLI):

Dati A, **b** e **c** interi, trovare un **x** a valori 0, 1 che soddisfa $A\mathbf{x} \geq \mathbf{b}$ e minimizza $\mathbf{c}^T\mathbf{x}$.

Suggerimento: procedere per riduzione dal problema di soddisfacibilità (SAT)