DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

Complessità computazionale

Introduzione

Scopo: Stimare lo sforzo computazionale per risolvere problemi di ottimizzazione

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

Perchè:

Conoscendo l'ordine di complessità di algoritmi alternativi si può selezionare quello più efficiente.

Problema, Istanza, Algoritmo

Problema: il caso generale

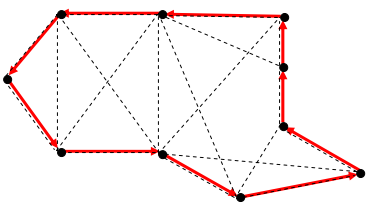
Esempio: Dato un grafo G, esiste in G un circuito hamiltoniano?

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.

Misuro i collegamenti con i tempi



Problema

Dato un grafo orientato $G = (N, A)$, con un costo $c_{ij} \in \mathbf{Z}$, $\forall (i,j) \in A$, determinare un circuito di costo minimo che visita esattamente una volta 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, …

Problema, Istanza, Algoritmo

Problema: il caso generale (Dato un grafo G, esiste in G un circuito hamiltoniano?)

Istanza: ogni caso particolare (Dato "questo" grafo G, esiste in G un circuito hamiltoniano?)

Algoritmo: procedura di calcolo in grado di risolvere ogni singola istanza di un problema (Dato un qualunque grafo G è in grado di dire se esiste oppure no un circuito hamiltoniano)

Complessità degli algoritmi

Scopo: Stimare l'onere computazionale per risolvere un dato problema P mediante gli algoritmi alternativi

$\Rightarrow$ selezionare l'algoritmo 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,

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

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

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

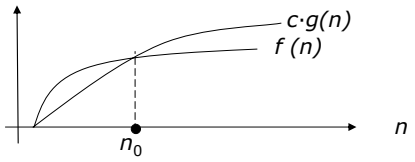
Ordine di complessità

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

numero di OE per risolvere istanza $I \le f(n) \quad \forall I$ con $|I| = n$

Complessità asintotica :

$f(n) = O(g(n))$ se $\exists c > 0$ tale che $f(n) \le c\, g(n)$, 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 $O(n^{10})$ non lo è molto in pratica!

Complessità computazionale di un **algoritmo**

è una funzione che ad ogni n associa il numero massimo di operazioni elementari necessarie a risolvere una istanza di "dimensione" n)

Tempo richiesto da algoritmi di varia complessità per risolvere problemi di dimensione *n* su un calcolatore che esegue 1.000.000 di operazioni elementari al secondo

Dimensione	Complessità computazionale				
n	n	n^2	n^5	2^n	3^n
10	0,00001"	0,0001"	0,1"	0,001"	0,059"
20	0,00002"	0,0004"	3,2"	1"	58'
30	0,00003"	0,0009"	24,3"	17,9'	6,5 a
40	0,00004"	0,0016"	1,7'	12,7 g	3855 s
50	0,00005"	0,0025"	5,2'	35,7 a	2*10^8 s

Tempo richiesto da algoritmi di varia complessità per risolvere problemi di dimensione *n* su un calcolatore che esegue 1.000.000 di operazioni elementari al secondo

Dimensione	Complessità computazionale				
n	n	n^2	n^5	2^n	3^n
10	0,00001"	0,0001"	0,1"	0,001"	0,059"
20	0,00002"	0,0004"	3,2"	1"	58'
30	0,00003"	0,0009"	24,3"	17,9'	6,5 a
40	0,00004"	0,0016"	1,7'	12,7 g	3855 s
50	0,00005"	0,0025"	5,2'	35,7 a	2*10^8 s

"buoni" "cattivi"

Algoritmi “buoni” e “cattivi”

Complessità “polinomiale” quando la funzione complessità è un polinomio od è limitata superiormente da un polinomio: n^2; 5n*log(n); ecc.

Complessità “esponenziale” negli altri casi: 3*2^n; n!; ecc.

Effetto di un computer 1000 volte più potente sulla dimensione massima dell’istanza risolubile in un’ora

Algoritmo di complessità computaz.	su un computer da 1.000.000 oper. elementari al secondo	su un computer da 1.000.000.000 oper. elementari al secondo
n	3,6 miliardi	3600 miliardi
n^3	60.000	1.897.000
n^5	82	324
2^n	32	42
3^n	20	26

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

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

Partendo dal flusso iniziale φ_0 di valore 0, si ha quindi una complessità totale $O(m^2 k_{max})$

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

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

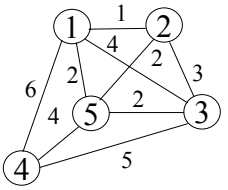
Modifica che lo rende $O(m^2 n)$:
Cercare cammino aumentante con un numero minimo di archi (Edmonds e Karp)

Esercizio

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

Istanza specificata dai valori di n e m, dalla struttura di $G = (N, E)$ e dai costi $c_e \; \forall \; e \in E$

$n = 5$



$$\text{dimensione istanza} \leq \lceil \log n \rceil + \lceil \log m \rceil + m \cdot \lceil \log \overline{c} \rceil + n^2$$

dove $\overline{c} = \max_{e \in E} c_e$

N.B.: dimensione $\geq n$

Complessità dei problemi

Scopo: Stimare la *difficoltà intrinseca* di un problema che corrisponde alla complessità del "miglior algoritmo che permetterebbe di risolverlo"

$\Rightarrow$ adottare l'approccio più adeguato

Un problema P è *polinomiale* (*facile*) se esiste un algoritmo polinomiale che lo risolve.

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"!

Teoria della *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 $\leq L$?)

Versione di riconoscimento è difficile

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

Classi di complessità

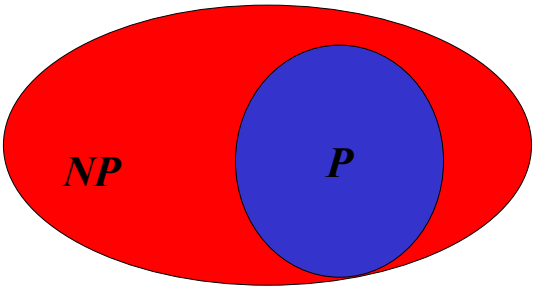
- ***P*** indica la classe dei problemi di riconoscimento che si possono risolvere in tempo polinomiale.

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

- ***NP*** indica la classe dei problemi di riconoscimento per i quali esiste una prova del fatto che un'istanza abbia risposta "si" che può essere verificata in tempo polinomiale.

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

Chiaramente $P \subseteq NP$



Congettura $P \subset NP$

Classi di problemi: Classe *P* e Classe *NP*

Classe ***P:*** insieme dei problemi per i quali esiste un algoritmo polinomiale.

Esempi: PL; Albero di supporto di costo minimo; Cammino minimo ecc.

Classe ***NP:*** insieme dei problemi risolubili in tempo polinomiale su una macchina di Turing non deterministica (cosa vuol dire?)

Classe *NP*

Un problema (di riconoscimento) sta nella classe ***NP*** quando le istanze che hanno "sì" come risposta corretta, possiedono un "certificato" che può essere "verificato" in tempo polinomiale.

N.B.1: "Verificato", non "trovato"!

N.B.2: Nulla si impone sulle istanze che hanno "no" come risposta corretta!

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 $P_2 \in$ ***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 ,
- 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, esiste un ciclo hamiltoniano di lunghezza $\leq L$?

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

$$P_1 \propto P_2$$

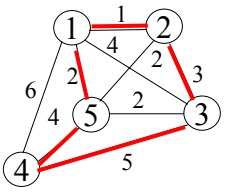
Riduzione polinomiale dal caso non orientato a quello orientato :

in tempo e spazio polinomiale

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

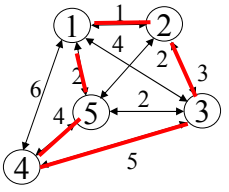
$G=(N,E)$

$L = 15$



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

$L' = 15$



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

Conseguenza:

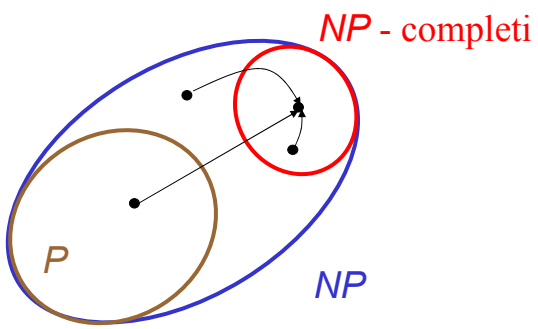
Se $P_1 \propto P_2$ e P_2 è risolubile mediante un algoritmo polinomiale, allora esiste anche un algoritmo polinomiale per P_1 ($P_2 \in$ ***P*** $\Rightarrow$ $P_1 \in$ ***P***)

Problemi *NP* – completi

Un problema P è ***NP*** – *completo* se e solo se

1) appartiene a ***NP***

2) ogni altro problema in ***NP*** è riducibile ad esso in tempo polinomiale ($P' \propto P$, $\forall$ P' $\in$ ***NP***)



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

Eventualità considerata molto improbabile

La ***NP*–completezza** è quindi un **forte indizio** di **difficoltà** intrinseca

cf. lunga lista di problemi ***NP***-completi per i quali non sono noti algoritmi polinomiali

Esempi di problemi *NP* – completi

• Dato $G = (N, E)$ non orientato con costi sugli archi e un intero L, esiste un ciclo hamiltoniano di lunghezza $\leq L$?

• 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 un sistema lineare $A\mathbf{x} \geq \mathbf{b}$ con coefficienti interi, esiste una soluzione $\mathbf{x}$ a valori 0, 1 ?

•….

Come mostrare la *NP* – completezza ?

Per mostrare che un problema P_2 è ***NP*** -completo "basta" dimostrare che un altro problema ***NP*** -completo P_1 si riduce polinomialmente a P_2 :

$P \propto P_1, \forall$ P $\in$ ***NP*** , e $P_1 \propto P_2$ implica per transitività che

$P \propto P_2, \forall$ P $\in$ ***NP***

Esempio

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

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

$P_2 \in$ ***NP*** e $P_1 \propto P_2$ con $P_1 \in$ ***NP*** -completo

Problemi *NP* – difficili

Un problema è ***NP*** – *difficile* se non appartiene a ***NP*** ma ogni altro problema in ***NP*** è riducibile ad esso in tempo polinomiale

Esempi

1) TSP poiché TSP-r (esiste un circuito hamiltoniano di lunghezza $\leq L$?) è *NP* – completo

2) Programmazione Lineare Intera (PLI):
Dati A, **b** e **c** interi, trovare un **x** a valori 0, 1 che soddisfa A **x** $\geq$ **b** e minimizza **c** **x**.

Primo problema dimostrato NP-completo (T. Cook) :

Soddifattibilità (SAT)

Dato un insieme di *m* clausole booleane $\mathbf{C}_1,\ldots,\mathbf{C}_m$ (disgiunzioni -- OR -- di variabili booleane o loro complementi), esiste un assegnamento di valori "vero" o "falso" alle variabili che rende vere tutte le clausole?

Esempio

$$C_1 = (y_1 \wedge y_2 \wedge y_3)$$
$$C_2 = (\overline{y_1} \wedge y_2)$$
$$C_3 = (y_2 \wedge \overline{y_3})$$

Assegnamento: y_1 = **Falso**, y_2 = **Vero**, y_3 = **Falso**

Per mostrare che un dato problema P_2 in *NP* è NP-completo "basta" dimostrare che un altro problema NP-completo P_1 si riduce polinomialmente a P_2 :

$\forall$ P $\in$ *NP* , $P \{ P_1$ e $P_1 \{ P_2$ implica per transitività che $P \{ P_2$, $\forall$ P $\in$ *NP*

Esempio Esistenza di una soluzione di un sistema lineare con variabili binarie (PLI)

- $\in$ *NP*
- $SAT \{$ PLI

$$C_1 = (y_1 \wedge y_2 \wedge y_3)$$
$$C_2 = (\overline{y_1} \wedge y_2)$$
$$C_3 = (y_2 \wedge \overline{y_3})$$

$\Leftrightarrow$

$$1 \le x_1 + x_2 + x_3$$
$$1 \le (1 - x_1) + x_3$$
$$1 \le x_1 + (1 - x_3)$$