CAP 2
ELEMENTI DI COMPLESSITÀ COMPUTAZIONALE

Classificazione dei *prob di ottimizzazione* in base alla **difficoltà intrinseca del prob**, *indipendentemente* dall'**algo**.

Obiettivi principali

- distinguere **complessità** del *prob vs algo*
- introdurre **prob di decisione vs ottimizzazione**
- definire le classi **P**, **NP**, **NPH**, **NPC**

COMPLESSITÀ INTRINSECA: SCHEDULING

Problema di scheduling (min,max) su macchina singola. Sequenziamento di job su una macchina per controllare il max ritardo rispetto alle due date.

$$C_{\sigma(1)} = p_{\sigma(1)},$$
$$C_{\sigma(k)} = C_{\sigma(k-1)} + p_{\sigma(k)}, \quad k = 2, \ldots, n.$$

$$L_{\max} \doteq \max_{j \in [n]} L_j \qquad L_j \doteq C_j - d_j.$$

Teorema (regola EDD – Earliest Due Date). Per il prob *1\\Lmax* esiste una *sol ottima* in cui i <u>job sono ordinati per due date crescenti</u>. $d_{\sigma(k)} \leq d_{\sigma(k+1)}$

➡️ *algo polinomiale*

➡️ *prob computazionalmente trattabile*

Realise Time. L'introduzione dei tempi di rilascio (*1\ri\Lmax*) vincola l'avvio dei job e rende non più ottima la regola EDD, <u>aumentando la compl intrinseca del prob</u>. **Non esistono** algo di compl polinomiale che lo risolvono, solo **branch-and-bound**.

PROBLEMI DI DECISIONE VS OTTIMIZZAZIONE

- **prob di decisione (PD)**: risposta binaria *(sì / no)*
- **prob di ottimizzazione (PO)**: ricerca migliore sol rispetto a una f obiettivo $\min_{x \in S} f(x)$

Legame tra PO e PD. Dato un PO, si definisce un PD scegliendo un valore k e chiedendo se esiste x in S tc f(x)<k.

PD→PO. Se si ha a disposizione un algo efficiente per PO, è possibile risolvere in modo efficiente PD

➡️ *se PD è difficile ⇒ PO non può essere facile*

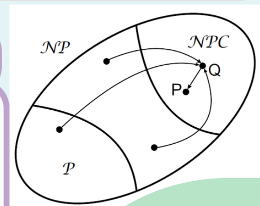
➡️ *per dimostrare che un prob di ottimizzazione è difficile, è sufficiente dimostrare che è difficile il corrispondente prob di decisione*

SCHEDULING CON RELEASE TIMES

La versione decisionale del prob di scheduling 1/rj/Lmax è **NP-completa**.

- *mostra che l'intrattabilità nasce con l'introduzione dei tempi di rilascio*
- *PO è NP-difficile*

TEORIA DELLA NP-COMPLETEZZA



Esiste una vasta classe di *prob di ottimizzazione* per cui non sono noti algo polinomiali.

Idea centrale. La **teoria della NP-completezza** mostra che molti di questi prob sono <u>equivalenti</u> dal punto di vista *computazionale*.

➡️ *se uno solo ammettesse un algo pol tutti i prob della classe lo ammetterebbero*

Conseguenza fondamentale. Decenni di ricerca senza successo suggeriscono che

- *tali algo probabilmente non esistono*
- *la difficoltà è intrinseca, non dovuta a modelli*

CLASSI NPH – NPC

Classe NPH. Prob P tc ogni prob in NP è riduc a P.

➡️ *PO + PD*

Classe NPC. Prob P tc

- P è in NP **e** P è NPH

➡️ *problemi più difficili in NP, tutti equivalenti*

➡️ *per dimostrare che un PD P è NPC, occorre dimostrare che P è in NP e un prob NP-completo Q può essere ridotto in tempo polinomiale a P*

Teorema di Cook. Il prob della soddisfacibilità booleana è NP-completo.

$$(A \text{ or } B) \text{ and } (\text{not}(A) \text{ or } C)$$

CLASSI P – NP

Classe P. <u>*Prob di decisione*</u> per cui esiste un algo di compl polinomiale: *il numero di passi è limitato superiormente da una funzione polinomiale.* L'**algo**

- *trova una soluzione*
- *ne verifica la correttezza*

Classe NP. *Prob di decisione* le cui istanze che hanno risposta positiva sono verificabili in tempo polinomiale. Esiste un certificato polinomiale su un ***calcolatore non deterministico***.

RIDUZIONE POLINOMIALE

Un prob P è **riducibile** a Q se *ogni istanza di P può essere trasformata in tempo polinomiale in un'istanza di Q* con la stessa risposta.

- se P è difficile e **P<Q** ⇒ Q non può essere facile
- *la compl di P non è maggiore della compl di trasformare P in Q e poi risolvere Q*

$$\text{compl}(P) \leq \text{compl}(Q) + \text{compl}(P \to Q)$$

IMPATTO CODIFICA (KNAPSACK)

Idea chiave. La compl dipende dalla codifica dell'input, non solo dal prob.

- *B* è **codificato in binario** ⇒ input di dimensione logB
- *O(nB)* è **esponenziale** nella dimensione dell'input

Pseudo-Polinomiale. L'algo, rispetto alla codifica binaria, ha *compl esponenziale*. Se si utilizzasse un **computer con una codifica unaria**, l'algo avrebbe *compl polinomiale*.