Complessità nei problemi di scheduling

1. Problemi di ottimizzazione combinatoria

- Un'istanza I di un problema di ottimizzazione combinatoria è la coppia (X, f), dove:
  - $X \subseteq \Omega$ è un *insieme finito* (soluzioni ammissibili)
  - f è una funzione di costo (obiettivo) $(f: \Omega \rightarrow R)$ che assegna un valore reale (costo) ad ogni $x \in \Omega$

  Il problema associato consiste nel determinare una soluzione ammissibile *ottima* $x^* \in X$ tale che $f(x^*) \leq f(x)$, $\forall\, x \in X$, e può essere rappresentato come

$$\begin{aligned} &\min f(x) \\ &\text{s.t.} \\ &x \in X \subseteq \Omega \end{aligned}$$

- Un problema di ottimizzazione combinatoria Π è l'insieme I di tutte le sue istanze

Complessità nei problemi di scheduling

2. Problemi e istanze

- Col termine *problema* ci si riferisce alla classe delle sue istanze (come ad esempio $1||\Sigma C_j$) e col termine *istanza* ci si riferisce a casi specifici (una volta assegnati i valori ai parametri del problema).

- Per il problema $1||\Sigma C_j$ è sufficiente conoscere i tempi di processamento dei job per specificare un'istanza.

- La *dimensione* di una istanza è determinata dai dati del problema e dallo *schema di codifica* utilizzato per rappresentare i dati; è pari al numero dei caratteri (secondo lo schema di codifica utilizzato) necessari per specificare l'istanza.

Complessità nei problemi di scheduling

3. Codifica unaria e binaria

- Un'istanza può essere rappresentata considerando *zeri e uni* (*codifica binaria*) o *solo uni* (*codifica unaria*).

- Ad esempio data un'istanza I di $1||\Sigma C_j$ specificata attraverso i tempi di processamento dei job

$$(2, 5, 2, 3, 5, 5, 8)$$

- Se la codifica utilizzata è quella *binaria* la lunghezza della stringa (cioè la dimensione dell'istanza) è data da

$$L_B(I) = \text{Length}(10, 101, 10, 11, 101, 101, 1000) = 19 + 7$$

- Se la codifica impiegata è quella *unaria*

$$L_U(I) = \text{Length}(11, 11111, 11, 111, 11111, 11111, 11111111) = 30 + 7$$

- Chiaramente la dimensione di una istanza codificata in modo unario è maggiore di quella codificata in modo binario.

- Di solito ci si riferisce ad una *codifica binaria*

Complessità nei problemi di scheduling

4. Complessità computazionale di un algoritmo

- Sia $\mathcal{A}$ un algoritmo che risolve il problema Π; siamo interessati a valutare l'efficienza di $\mathcal{A}$ rispetto al suo tempo computazionale.

- La funzione di *tempo di calcolo* $t(n)$ di un algoritmo $\mathcal{A}$ esprime il massimo (*worst case*) numero di operazioni eseguite da $\mathcal{A}$ per risolvere una qualsiasi istanza I di Π di dimensione $n = L_B(I)$.

- Un algoritmo $\mathcal{A}$ ha *complessità tempo polinomiale* se la sua funzione di tempo di calcolo $t(n)$ è $O(p(n))$ dove $p(n)$ è un polinomio in n cioè

$$\exists\, c,\ n_0 > 0 : t(n) \le c \cdot p(n),\quad \forall\ n \ge n_0$$

- Altrimenti, diremo che $\mathcal{A}$ ha *complessità tempo esponenziale*.

Esempio

Se $t(n) = 500 + 100\,n^2 + 5\,n^3 \quad \Rightarrow \quad O(n^3)$

$O(n^3)$ e' la *complessità computazionale di* $\mathcal{A}$

- In particolare, diremo che $\mathcal{A}$ ha *complessità tempo pseudo-polinomiale* se la sua funzione tempo di calcolo $t(n)$ è $O(p(L_B[I], \mathrm{MAX}[I]))$, dove $L_B[I] = n$ e $\mathrm{MAX}[I]$ è il massimo intero nell'istanza I.
(N.B.: $\mathcal{A}$ è polinomiale se si fa uso di codifica unaria)

Complessità nei problemi di scheduling

5. Complessità di un problema di decisione

- Un *problema* che può essere risolto da un algoritmo tempo *polinomiale* è detto *trattabile*.

- Ci sono però problemi per i quali *non sono noti algoritmi* di risoluzione *polinomiali*.

- Per un tale problema sarebbe utile poter affermare che è *intrattabile*, cioè *non esistono* affatto *algoritmi* di risoluzione *polinomiali* che lo risolvono.

- Volendo caratterizzare formalmente tale aspetto ci si avvale della nozione di *problema NP-completo*.

- La *classe dei problemi NP-completi* ha le seguenti *proprietà*:
  1) *Nessun problema NP-completo* può essere risolto da un algoritmo *polinomiale noto*.
  2) Se *esistesse* un algoritmo *polinomiale* per un *problema NP-completo* allora ci sarebbe un algoritmo *polinomiale* per *ogni problema NP-completo*.

- Sulla base di quanto detto *si congettura che non ci sia alcun algoritmo polinomiale per un problema NP-completo*.

Complessità nei problemi di scheduling

5. Complessità di un problema di decisione (continua)

- Pertanto è *ampiamente ritenuto che un problema NP-completo sia intrattabile* dal punto di vista computazionale.

Ne segue che:

- Ogni *algoritmo* che risolve un *problema NP-completo* richiede, nel caso peggiore, una quantità di *tempo esponenziale* nella dimensione dell'istanza. Quindi, la sua utilità è confinata a piccole istanze.

- Pertanto, quando si affronta un *problema* e *non* si riesce a trovare un *algoritmo polinomiale* che lo risolve, si può cercare di dimostrare che il problema è *NP-completo*.

- Una volta che il *problema* è noto essere *NP-completo*, l'analisi può proseguire secondo *tre linee principali*:

  1) Studio di opportuni *rilassamenti del problema*.

  2) Definizione di *algoritmi euristici* e di *approssimazione*.

  3) Sviluppo di *algoritmi esatti*.

Complessità nei problemi di scheduling

6. Problemi di ottimizzazione come problemi di decisione

- Ogni problema Π risulta definito dall'insieme $\mathcal{I}$ delle sue istanze I.

- Ogni istanza $I \in \mathcal{I}$ può essere definita come una coppia (X, f) dove $X \subseteq \Omega$ è l'insieme delle soluzioni ammissibili e $f: \Omega \rightarrow R$ è una funzione di costo.

- *Problema di ottimizzazione*

  Data un'istanza $I = (X, f)$ di Π determinare un $x^* \in X$ tale che $f(x^*)$ sia di valore ottimo (ad es. minimo).

- *Problema di decisione*

  Data un'istanza $I = (X, f)$ di Π e un valore reale L, esiste una soluzione ammissibile $x^* \in X$ tale che $f(x^*) \leq L$?

  Le istanze $\mathcal{I}$ di un problema di decisione possono essere partizionate in istanze *Yes* $\mathcal{I}_Y$ e istanze *No* $\mathcal{I}_N$.

Complessità nei problemi di scheduling

7. Classi di complessità: P e NP

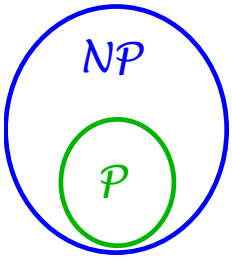
- La classe $\mathcal{P}$ contiene tutti i *problemi decisionali* risolvibili da un algoritmo tempo *polinomiale*.

- La classe $\mathcal{NP}$ contiene tutti i *problemi decisionali* le cui *istanze Yes*, dato un *certificato conciso*, possono essere *verificate* da un algoritmo *tempo polinomiale* nella dimensione dell'istanza e del certificato.

- In pratica, *la classe $\mathcal{NP}$ contiene tutti i problemi ingegneristici di progettazione ottima di "oggetti"*. Per tali problemi, infatti, la soluzione è esprimibile in modo conciso e costituisce un certificato conciso per la prova di appartenenza del problema alla classe $\mathcal{NP}$.

- $\mathcal{P} \subseteq \mathcal{NP}$; infatti l'algoritmo che risolve un problema $\pi \in \mathcal{P}$ può essere usato come algoritmo di certificazione e la sequenza di operazioni eseguite come certificato.

- *Non* è stato *dimostrato* che $\mathcal{P} \equiv \mathcal{NP}$, anzi si *congettura* che $\mathcal{P} \subset \mathcal{NP}$.



Complessità nei problemi di scheduling

8. Classi di complessità: NP-completezza

Definizione

- Un problema Π_1 è *riducibile* (*polinomialmente*) a un problema Π_2 ($\Pi_1 \propto \Pi_2$) se esiste una funzione g computabile in tempo polinomiale che data l'istanza x di Π_1 la trasforma nell'istanza $y = g(x)$ di Π_2, tale che x è una istanza *Yes* di Π_1 *se e solo se* y è una istanza *Yes* di Π_2.

- La funzione g con le suddette proprietà che associa ad x una tale y è detta *trasformazione polinomiale*.

- L'operazione di riduzione $\propto$ e' una operazione *transitiva*

$$(\Pi_1 \propto \Pi_2) \text{ e } (\Pi_2 \propto \Pi_3) \Rightarrow \Pi_1 \propto \Pi_3$$

Definizione: Problema $\mathcal{NP}$-completo

Un problema $\Pi \in \mathcal{NP}$ è detto *$\mathcal{NP}$-completo* se tutti gli altri problemi in $\mathcal{NP}$ sono riducibili a Π.

- In base alla proprietà transitiva, un problema $\Pi \in \mathcal{NP}$ è *$\mathcal{NP}$-completo* se esiste un problema *$\mathcal{NP}$-completo* Π' che si riduce a Π ($\Pi' \propto \Pi$).

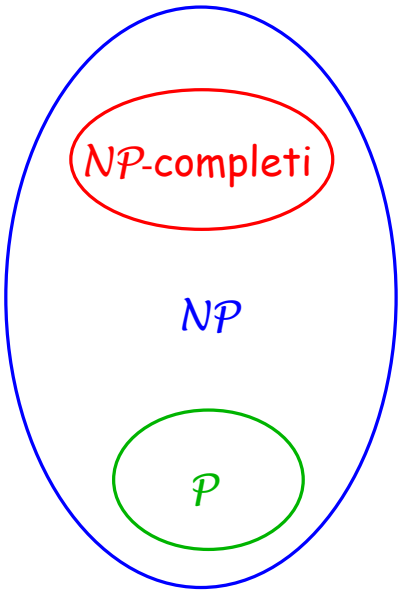
Complessità nei problemi di scheduling

8. Classi di complessità: NP-completezza (continua)

- Un problema *NP-completo* è *non più semplice* (dal punto di vista computazionale) di ogni altro problema in $\mathcal{NP}$.

- D'altra parte, *se* esistesse un problema *NP-completo* risolvibile in *tempo polinomiale*, lo sarebbero *tutti* i problemi $\mathcal{NP}$, utilizzando le trasformazioni polinomiali da questi al problema *NP-completo* e risolvendo poi polinomialmente quest'ultimo.

- In base alla *congettura* $\mathcal{P} \subset \mathcal{NP}$ si può affermare che per un problema *NP-completo* *non* esistono algoritmi *polinomiali* di risoluzione, cioè $\mathcal{P} \cap \mathcal{NP}\text{-completi} = \varnothing$.



- La versione di *ottimizzazione* di un *problema NP-completo* è detta *NP-hard*.

Complessità nei problemi di scheduling

9. Principali problemi NP-completi

- Il "primo" problema *NP-completo* (Cook, 1971)

  SATISFIABILITY (SAT)

  Istanza: Un insieme di var. binarie, e una collezione di clausole definite sulle variabili.

  Domanda: Esiste un assegnamento di valori per le variabili per cui ogni clausola è soddisfatta ?

- I *sei problemi NP-completi* "di *base*" e la sequenza di trasformazioni usate nella prova di NP-completezza (Karp, 1972)

  3 SATISFIABILITY (3-SAT)

  Istanza: Un insieme di var. binarie, e una collezione di clausole composte esattamente da 3 letterali.

  Domanda: Esiste un assegnamento di valori per le variabili per cui ogni clausola è soddisfatta?

  3-DIMENSIONAL MATCHING (3-DM)

  Istanza: Un insieme $M \subseteq W \times X \times Y$, dove W, X e Y sono insiemi disgiunti di q elementi ciascuno.

  Domanda: M contiene un matching, cioe un sottoinsieme $M' \subseteq M$ tale che $|M'| = q$, e che non esistono due elementi di M' che hanno almeno una stessa coordinata ?

Complessità nei problemi di scheduling

9. Principali problemi NP-completi (continua)

PARTITION

Istanza: Insieme $S = \{a_1, \ldots, a_n\}$ di interi con $\Sigma_i a_i = 2B$.

Domanda: Esiste una partizione di S in S_1 e S_2, tale che $\Sigma_{a_i \in S_1} a_i = \Sigma_{a_i \in S_2} a_i = B$?

VERTEX COVER (VC)

Istanza: Grafo $G = (V, E)$ e un intero positivo $k \leq |V|$.

Domanda: Esiste un vertex cover di cardinalità non maggiore di k per G, cioè un sottoinsieme $V' \subseteq V$ tale che $|V'| \leq k$, e per ogni $(u, v) \in E$ almeno u o v appartiene a V' ?

CLIQUE

Istanza: Grafo $G = (V, E)$ e un intero positivo $q \leq |V|$.

Domanda: Esiste in G una clique di cardinalità non inferiore a q, cioè un sottoinsieme $V' \subseteq V$ tale che $|V'| \geq q$, e per ogni coppia u, v in V' si ha $(u, v) \in E$?

HAMILTONIAN CIRCUIT (HC)

Istanza: Grafo $G = (V, E)$

Domanda: Esiste in G un circuito hamiltoniano ?

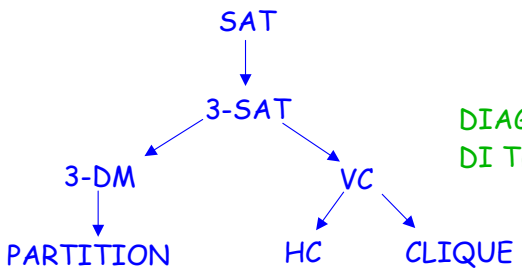


DIAGRAMMA DELLA SEQUENZA DI TRASFORMAZIONI USATE

Complessità nei problemi di scheduling

10. Problemi numerici e NP-completi in senso forte

Definizione

- Un problema Π è detto "*numerico*" se *non esiste* alcun polinomio $p(\bullet)$ tale che $\mathrm{MAX}[I] \leq p(L_B[I])$ per ogni istanza I di Π.

- Il *PARTITION* è un *problema numerico*. Gli altri sei problemi principali definiti prima non sono numerici.

Osservazione 1

Se $\Pi \in \mathcal{NP}$-completi *non* è un problema *numerico*, allora Π *non* può essere risolto da un *algoritmo pseudo-polinomiale*, a meno che $\mathcal{P} = \mathcal{NP}$.

Quindi, gli unici problemi $\mathcal{NP}$-completi candidati ad essere risolti da un algoritmo *pseudo-polinomiale* sono i problemi *numerici*

- Dato un problema Π e un polinomio $p(\bullet)$ sia Π_p il problema ristretto alle sole istanze I tali che $\mathrm{MAX}[I] \leq p(L_B[I])$.
  (N.B.: Π_p *non* è un problema *numerico*)

- Se Π è risolvibile da un algoritmo *pseudo-polinomiale*, allora Π_p è risolvibile in tempo *polinomiale*.

Complessità nei problemi di scheduling

10. Problemi numerici e NP-completi in senso forte (continua)

Definizione: Problema $\mathcal{NP}$-completo in senso forte

Un problema Π è *$\mathcal{NP}$-completo in senso forte* se $\Pi \in \mathcal{NP}$ ed esiste un polinomio $p(\bullet)$ tale che $\Pi_p \in \mathcal{NP}$-completi.

- Se un problema Π è *$\mathcal{NP}$-completo* e *non* è *numerico*, Π è automaticamente *$\mathcal{NP}$-completo in senso forte*

- A questo punto possiamo generalizzare l'Osservaz. 1.

Osservazione 1

Se Π è *$\mathcal{NP}$-completo in senso forte* allora Π *non* può essere risolto da un *algoritmo pseudo-polinomiale*, a meno che $\mathcal{P} = \mathcal{NP}$.

- Il *PARTITION* è un problema *numerico* che può essere risolto in tempo *pseudo-polinomiale* $O(nB)$ tramite un algoritmo di programmazione dinamica e quindi è *$\mathcal{NP}$-completo ma non in senso forte*.

- Un problema *numerico* e *$\mathcal{NP}$-completo in senso forte* è

  3-PARTITION

  Istanza: Un insieme $S = \{a_1, \ldots, a_{3z}\}$ di $3z$ interi e un intero B, tali che $B/4 < a_i < B/2$ e $\Sigma_{a_i \in S} a_i = zB$

  Domanda: Esiste una partizione di S in z sottoinsiemi $S_1, \ldots, S_z$ di 3 elementi ciascuno tali che $\Sigma_{a_i \in S_j} a_i = B$, per $j = 1, \ldots, z$?

Complessità nei problemi di scheduling

11. Come provare la NP-completezza di un problema

- La maggior parte delle *prove di $\mathcal{NP}$-completezza* sono *basate* sul concetto di *trasformazione polinomiale* e sulla sua proprietà transitiva.

- Un problema $\Pi \in \mathcal{NP}$, per il quale la complessità viene stabilita applicando una *trasformazione polinomiale* dal *PARTITION* a Π, è *$\mathcal{NP}$-completo*, ma non possiamo affermare che lo sia in senso forte.

- Per dimostrare che un problema Π è invece *$\mathcal{NP}$-completo in senso forte* occorre mostrare che
  - che esiste un polinomio $p(\bullet)$ per cui Π_p è *$\mathcal{NP}$-completo*, oppure che
  - $\Pi \in \mathcal{NP}$ ed esiste un problema *$\mathcal{NP}$-completo in senso forte* Π' che si *riduce pseudo-polinomialmente* a Π.

- La riduzione pseudo-polinomiale si realizza tramite una funzione g detta *trasformazione pseudo-polinomiale* la cui definizione estende quella di trasformazione polinomiale.

Complessità nei problemi di scheduling

11. Come provare la NP-completezza di un problema

- Una *trasformazione pseudo-polinomiale* da Π_1 a Π_2 è una funzione g che data l'istanza x di Π_1 la trasforma nell'istanza $y = g(x)$ di Π_2 tale che:
  a) x è una istanza *Yes* di Π_1 *se e solo se* y è una istanza *Yes* di Π_2;
  b) g è computabile in tempo pseudo-polinomiale;
  c) y non è sostanzialmente più corta di x;
  d) MAX[y] è al più pseudo-polinamiale rispetto a x (o MAX[x]).

- Possiamo notare che le prime tre condizioni sono senz'altro verificate anche da tutte le trasformazioni polinomiali usuali e che la condizione d) sostanzialmente richiede che la magnitudo del più grande valore numerico che compare nell'istanza costruita attraverso la trasformazione non cresca esponenzialmente rispetto alla lunghezza o al massimo valore numerico dell'istanza di partenza.

Ad esempio quindi

- Un problema Π, per il quale la complessità viene stabilita applicando una *trasformazione polinomiale* (che rispetta la condizione d) dal *3-PARTITION* a Π, è *NP-completo in senso forte*.

Complessità nei problemi di scheduling

12. Prove di NP-completezza per problemi di scheduling

Esempio 1

Consideriamo il problema di scheduling $1|\, d_j = d \,|\, \Sigma w_j U_j$ di n job con *stesse due date* su *una macchina* con l'obiettivo di *minimizzare il numero pesato dei job in ritardo* ($U_j = 1$ se $C_j > d_j$, e $U_j = 0$ altrimenti).

Formuliamo il problema matematicamente considerando la variabile

$$x_i = \begin{cases} 0 & \text{se il job i è in ritardo} \\ 1 & \text{altrimenti} \end{cases}$$

$$\max \sum_{i=1}^{n} w_i x_i$$

s.t.

$$\sum_{i=1}^{n} x_i p_i \le d$$

$$x_i \in \{0, 1\},\ i = 1, \ldots, n$$

dove p_i è il tempo di processamento del job i.

N.B.: $\Sigma_i w_i U_i = \Sigma_i w_i (1 - x_i)$,

Pertanto, $1|\, d_i{=}d \,|\, \Sigma w_i U_i$ equivale al *KNAPSACK BINARIO*.

Complessità nei problemi di scheduling

12. Prove di NP-completezza per problemi di scheduling (continua)

Trasformazione (per restrizione) di *PARTITION* al *KNAPSACK BINARIO*

PARTITION

Istanza: Insieme $S = \{a_1, \ldots, a_n\}$ di interi con $\Sigma_i a_i = 2B$.

Domanda: Esiste una partizione di S in S_1 e S_2, tali che $\Sigma_{a_i \in S_1} a_i = \Sigma_{a_i \in S_2} a_i = B$?

PARTITION si trasforma al seguente caso speciale del *KNAPSACK BINARIO* in cui

$$p_i = a_i,\ w_i = a_i,\ d = B$$

e ci si chiede se $\Sigma_i w_i x_i \geq B$

Conclusioni

- È facile mostrare che l'istanza del *PARTITION* è *yes* se e solo se la relativa istanza del *KNAPSACK BINARIO* (nella sua versione decisionale) è *yes*
- *PARTITION* $\propto$ *KNAPSACK BINARIO*, pertanto se il *KNAPSACK BINARIO* fosse risolvibile in tempo polinomiale lo sarebbe anche *PARTITION*
- Poiché *PARTITION* è *NP-completo* possiamo concludere che lo è *anche* *KNAPSACK BINARIO* e quindi $1|d_i=d|\Sigma w_i U_i$ è *NP-hard*.
- *Non* è *NP-hard in senso forte* perché può essere risolto in tempo $O(nd)$ mediante un algoritmo di programmazione dinamica.

Complessità nei problemi di scheduling

12. Prove di NP-completezza per problemi di scheduling (continua)

Esempio 2

$O3||C_{max}$: min. C_{max} in *open-shop* scheduling su 3 macchine.

Trasformazione da *PARTITION* a $O3||C_{max}$

PARTITION

Istanza: $S = \{a_1, \ldots, a_z\}$ insieme di z interi con $\Sigma_i a_i = 2B$.

Domanda: Esiste una partizione di S in S_1 e S_2, tali che $\Sigma_{a_i \in S_1} a_i = \Sigma_{a_i \in S_2} a_i = B$?

Consideriamo $3z+1$ job con i seguenti tempi di processamento per le operazioni:

$p_{1,i} = a_i,$	$p_{2,i} = 0,$	$p_{3,i} = 0,$	$1 \le i \le z$
$p_{1,z+i} = 0,$	$p_{2,z+i} = a_i,$	$p_{3,z+i} = 0,$	$z+1 \le z+i \le 2z$
$p_{1,2z+i} = 0,$	$p_{2,2z+i} = 0,$	$p_{3,2z+i} = a_i,$	$2z+1 \le 2z+i \le 3z$
$p_{1,3z+1} = B,$	$p_{2,3z+1} = B,$	$p_{3,\,3z+1} = B.$	

PARTITION ha soluzione "yes" se e solo se $C^*_{max} = \gamma = 3B$.

M_1	S_1	$\{3z+1\}$	S_2
M_2	$\{3z+1\}$	$(S_1 \cup S_2)$	
M_3	$(S_1 \cup S_2)$		$\{3z+1\}$

$0 \qquad B \qquad 2B \qquad \gamma = 3B$

Nec.: Se *PARTITION ha soluzione "yes"* allora per costruzione (vedi figura) $C^*_{max} = \gamma$.

Suff.: Se $C^*_{max} = \gamma$ il job $(3z+1)$-simo è schedulato senza interruzione e i job $1 \le i \le z$ (o i job $z+1 \le z+i \le 2z$ o i job $2z+1 \le 2z+i \le 3z$) sono partizionati in S_1 e S_2 il che comporta che *PARTITION ha soluzione "yes"* .

Complessità nei problemi di scheduling

12. Prove di NP-completezza per problemi di scheduling (continua)

Esempio 3

$F3||C_{max}$: min. C_{max} in *flow-shop* scheduling su 3 macchine.

Trasformazione da *3-PARTITION* a $F3||C_{max}$

3-PARTITION

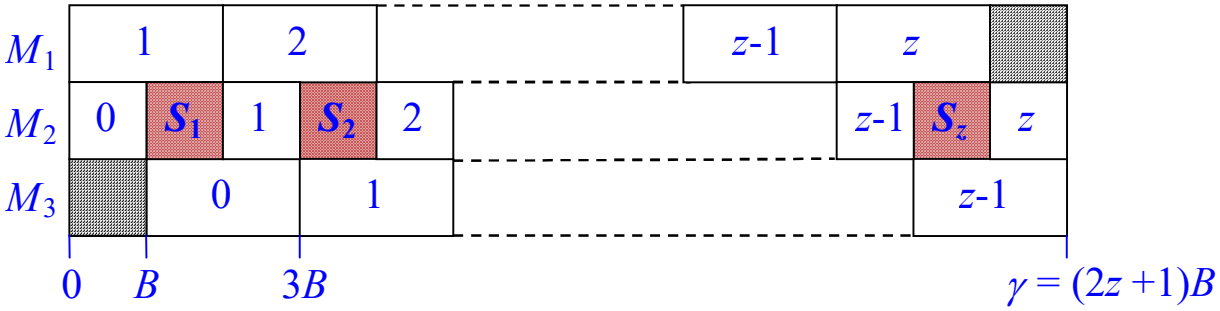
Istanza: Un insieme $S = \{a_1, \ldots, a_{3z}\}$ di $3z$ interi e un intero B, tali che $B/4 < a_i < B/2$ e $\Sigma_{a_i \in S} a_i = zB$

Domanda: Esiste una partizione di S in z sottoinsiemi $S_1, \ldots, S_z$ di 3 elementi ciascuno tali che $\Sigma_{a_i \in S_j} a_i = B$, per $j=1,\ldots, z$?

Consideriamo $4z+1$ job con tempi di process. per le operazioni:

$p_{1,0} = 0$, $p_{2,0} = B$, $p_{3,0} = 2B$,
$p_{1,j} = 2B$, $p_{2,j} = B$, $p_{3,j} = 2B$, $j = 1, \ldots, z-1$
$p_{1,z} = 2B$, $p_{2,z} = B$, $p_{3,z} = 0$,
$p_{1,z+i} = 0$, $p_{2,z+i} = a_i$, $p_{3,z+i} = 0$, $i = 1, \ldots, 3z$.

3-PARTITION ha soluz. "yes" se e solo se $C^*_{max} = \gamma = (2z+1)B$.



N.B.: $C^*_{max} = \gamma$ se i primi $z+1$ job sono schedulati secondo l'ordine $0, 1, \ldots, z$ lasciando z intervalli disgiunti su M_2 ciascuno lungo B dove poter schedulare i job $\{z+1, \ldots, 4z\}$.

Nec.: Se *3-PART. ha sol. "yes"* i job $\{z+1, \ldots, 4z\}$ possono essere schedulati su M_2 in tali intervalli, quindi $C^*_{max} = \gamma$.

Suff.: Se $C^*_{max} = \gamma$ i job $\{z+1, \ldots, 4z\}$ sono schedulati su M_2 in tali intervalli, implicando che *3-PART. ha sol. "yes"*.

Complessità nei problemi di scheduling

12. Prove di NP-completezza per problemi di scheduling (continua)

Alcune conclusioni

- *PARTITION* $\propto O3||C_{\max}$, pertanto se $O3||C_{\max}$ fosse risolvibile in tempo polinomiale lo sarebbe anche *PARTITION*.
- Poiché è noto che *PARTITION* è *NP-completo* (ma *non in senso forte*) si può solo concludere che $O3||C_{\max}$ è *NP-hard*.
- *3-PARTITION* $\propto F3||C_{\max}$, e poiché è noto che *3-PARTITION* è *NP-completo in senso forte*, possiamo concludere che $F3||C_{\max}$ è *NP-hard in senso forte*.

Alcune considerazioni finali

- Per *dimostrare* che un *problema è NP-hard* occorre *trovare* un *problema NP-completo* che sia *riducibile polinomialmente* al problema analizzato.
- Determinare una trasformazione polinomiale da un problema ad un altro non è un processo semplice
- Per dimostrare che un *problema è trattabile* occorre dimostrare l'esistenza di un *algoritmo tempo polinomiale* che lo risolve.

Complessità nei problemi di scheduling

13. Relazioni di complessità tra problemi di scheduling

- Spesso un algoritmo per un problema di scheduling può essere applicato ad altri problemi di scheduling.

  Per esempio

  $$1||\sum C_j \text{ è un caso speciale di } 1||\sum w_j C_j$$

  Un algoritmo che risolve $1||\sum w_j C_j$ *può essere usato anche per risolvere* $1||\sum C_j$

- Nella terminologia della complessità diremo che $1||\sum C_j$ *si riduce a* $1||\sum w_j C_j$.

  Utilizzando il formalismo utilizzato per le trasformazioni polinomiali diremo che

  $$1||\sum C_j \propto 1||\sum w_j C_j \ (1||\sum C_j \rightarrow 1||\sum w_j C_j)$$

- Sulla base di ciò, può ad esempio stabilirsi la seguente catena di riduzioni

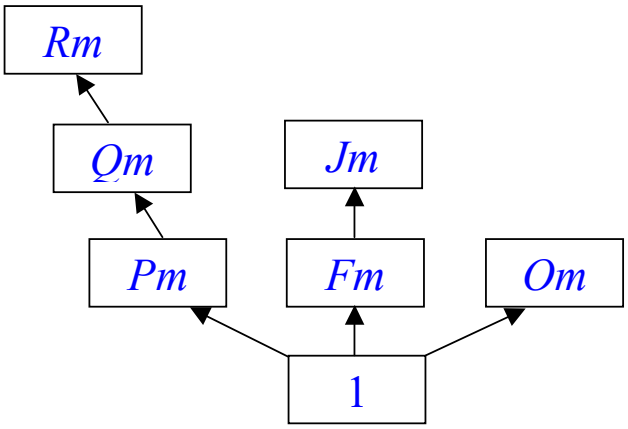
  $$1||\sum C_j \propto 1||\sum w_j C_j \propto Pm||\sum w_j C_j \propto Qm|\text{prec}|\sum w_j C_j$$

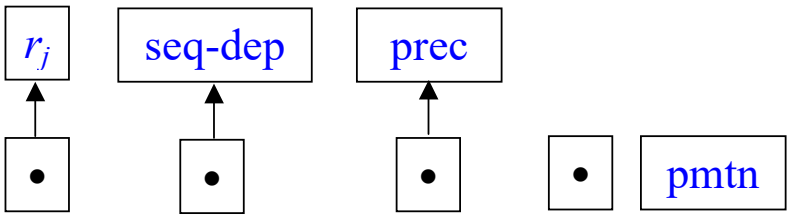
- Ovviamente vi sono molti problemi che non sono comparabili tra di loro, ad esempio $Pm||\sum w_j T_j$ e $Jm||C_{\max}$.

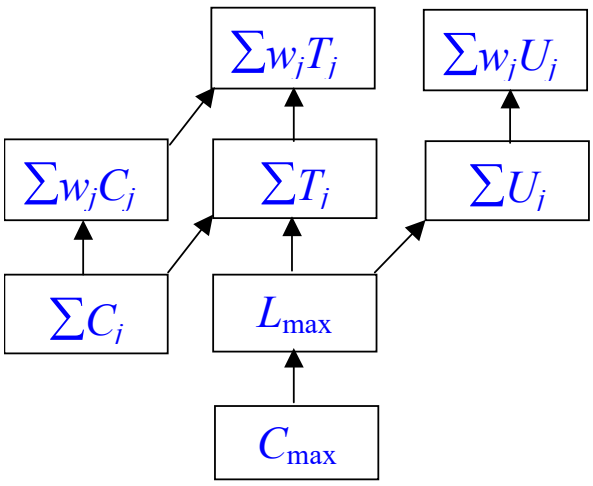
Complessità nei problemi di scheduling

13. Relazioni di complessità tra problemi di scheduling (continua)

- Esempi di gerarchie di complessità tra problemi di scheduling, basate per tipologia di *macchine*, *task*, *obiettivi*.







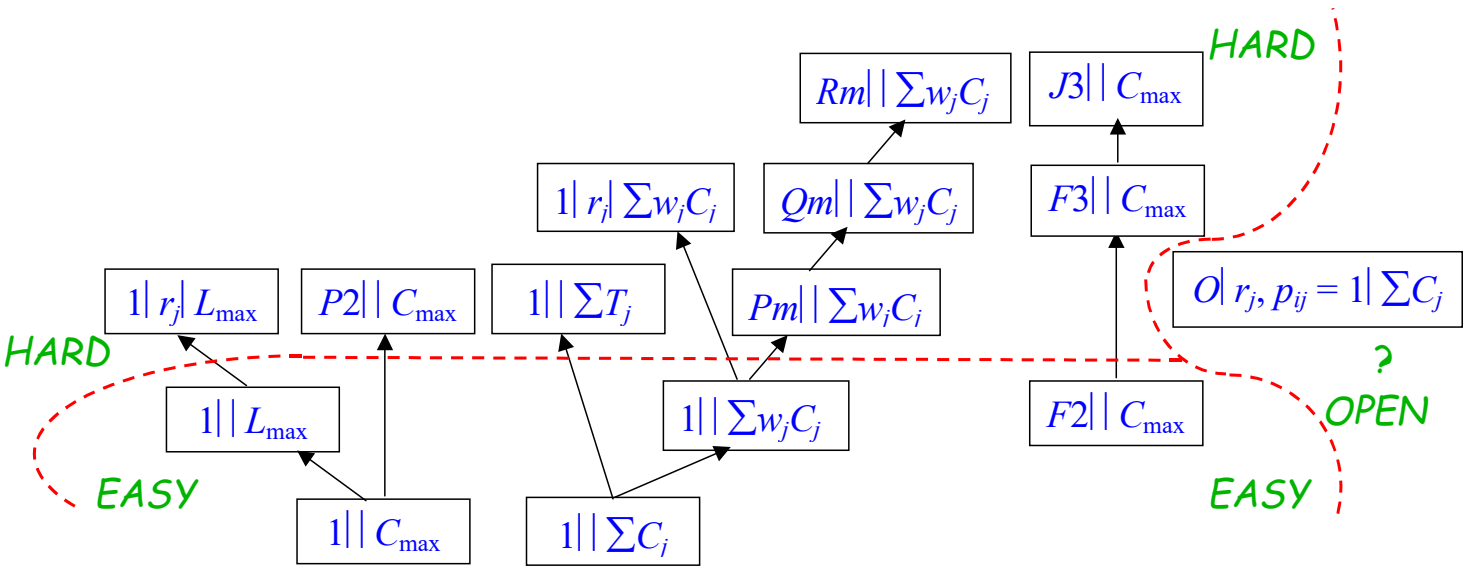
Complessità nei problemi di scheduling

13. Relazioni di complessità tra problemi di scheduling (continua)

- Per quanto detto circa la definizione di NP-completezza risulta che se $A \propto B$ e A è noto essere *NP-completo*, allora anche B è *NP-completo*.
- Ad esempio

$$F3||C_{\max} \rightarrow J3||C_{\max}$$

- Pertanto, noto che $F3||C_{\max}$ è *NP-hard (in senso forte)* lo è anche $J3||C_{\max}$.
- Esempi



- La *maggior parte* dei problemi di scheduling è *NP-hard*. Fra gli oltre 4000 problemi identificati solo poco più di 400 sono *trattabili*, mentre per circa 300 ancora *non se ne conosce la complessità*.