Algoritmi d'Approssimazione per Problemi Difficili

Algoritmi 2 – Lezione 12

A. Monti
Sapienza Università di Roma

Corso: Algoritmi 2

Indice

Il Problema della Minimizzazione del Makespan

Vogliamo organizzare l'esecuzione di una serie di lavori su un insieme di macchine e completare tutti i lavori nel minor tempo possibile.

- Più precisamente:
  Date m macchine identiche e n lavori, ognuno con un tempo di esecuzione p_j, assegnare ogni lavoro a una macchina per minimizzare il **makespan** ($C_{\max}$), ovvero il tempo di completamento dell'ultimo lavoro su tutte le macchine.

- Ad esempio, per $\mathit{lavori} = [11, 10, 8, 7, 6, 5]$ e $\mathit{macchine} = 3$ il makespan è 16 e si ottiene con questa distribuzione tripartizione

$$[[11, 5], [10, 6], [8, 7]]$$

- Per evitare casi banali si assume $2 \leq m < n$.

Makespan: un algoritmo 2-approssimante

- Questo problema sembra essere un problema difficile. Non si conosce un algoritmo in grado di trovare la soluzione ottima in un tempo "ragionevole" (polinomiale) al crescere delle dimensioni del problema (numero di lavori e macchine).
- Si ricorre quindi a euristiche che non garantiscono di trovare la soluzione ottima, ma trovano soluzioni di buona qualità in tempi rapidi.

Una semplice **euristica greedy** è la seguente:

1. Considera i lavori in un ordine qualsiasi.
2. Per ogni lavoro della lista,
   - assegnalo alla macchina che al momento ha il carico di lavoro totale minore.

Teorema di Approssimazione

- **Teorema:** L'euristica greedy ha un rapporto di approssimazione strettamente minore di 2, ovvero:

$$\frac{C_{\max}}{C^*_{\max}} < 2$$

dove:

- $C_{\max}$ è il makespan prodotto dall'algoritmo;
- $C^*_{\max}$ è il makespan ottimale.

Dettagli della Dimostrazione

- Sia k il lavoro che termina per ultimo e t_k il tempo in cui inizia.
- Poiché k termina per ultimo:

$$C_{\max} = t_k + p_k$$

- Sia S somma dei carichi di lavoro, ovvero $S = \sum_{i=1}^{n} p_i$. Al momento dell'assegnazione di k, tutte le macchine avevano carico almeno t_k (altrimenti k sarebbe andato su una macchina meno carica).

$$\Rightarrow S > m \cdot t_k \quad \Rightarrow \quad t_k < \frac{S}{m}$$

Sostituendo in $C_{\max}$: $\quad C_{\max} = t_k + p_k < \frac{S}{m} + p_k$

Dato che $\frac{S}{m} \leq C^*_{\max}$ e $p_k \leq C^*_{\max}$ otteniamo:

$$C_{\max} < C^*_{\max} + C^*_{\max} = 2 \cdot C^*_{\max}$$

Conclusione: $\frac{C_{\max}}{C^*_{\max}} < 2 \quad \Rightarrow \quad$ rapporto di approssimazione < 2. $\square$

Limite superiore per Greedy

Il caso peggiore si ha quando un unico, lunghissimo lavoro viene processato per ultimo, dopo che tanti piccoli lavori hanno già occupato le macchine. Consideriamo un'istanza con i seguenti parametri:

- **Numero di Macchine:** m
- **Lista di Lavori:** Un totale di $m(m-1)+1$ lavori, presentati nel seguente ordine: $m(m-1)$ lavori di durata 1, seguiti da un singolo lavoro di durata m (la lista è: $[1, 1, \ldots, 1, m]$).

Su quest'istanza proveremo che:

- **Il valore $C_{\max}$ della Soluzione Greedy** è: $C_{\max} = 2m - 1$
- **Il valore $C^*_{\max}$ della Soluzione Ottima** è $C^*_{\max} = m$.

Il rapporto tra il risultato dell'euristica e quello ottimo per questa istanza è:

$$\frac{C_{\max}}{C^*_{\max}} = \frac{2m-1}{m} = 2 - \frac{1}{m}$$

Al crescere del numero di macchine m, la frazione $\frac{1}{m}$ tende a zero e il rapporto tende a 2.

Questo esempio mostra che esistono istanze per cui il rapporto di approssimazione dell'euristica greedy è arbitrariamente vicino a 2, dimostrando che il bound di approssimazione non può essere migliorato.

Limite superiore per Greedy (2 / 2)

Analizziamo in dettaglio cosa accade per quest'istanza:

- **Il Valore $C_{\max}$ della Soluzione Greedy**. L'algoritmo processa la lista nell'ordine dato:
  - **Assegnazione dei lavori corti:** I primi $m(m-1)$ lavori di durata 1 vengono distribuiti in modo perfettamente bilanciato tra le m macchine. Ogni macchina riceve $m-1$ lavori, raggiungendo un carico di lavoro identico e pari a $m-1$.
  - **Assegnazione del lavoro lungo:** Arriva l'ultimo lavoro, di durata m. L'algoritmo lo assegna a una delle macchine (che hanno tutte lo stesso carico minimo). Il carico di questa macchina diventa:

$$C_{\max} = (\text{carico }) + (\text{nuovo lavoro}) = (m-1) + m = 2m-1$$

  Il makespan prodotto dall'euristica è quindi $2m-1$.

- **Il Valore $C^*_{\max}$ della Soluzione Ottima**. Una soluzione ottimale gestisce il lavoro lungo in modo strategico:
  - Assegna il lavoro di durata m a una macchina, M_1.
  - Distribuisce i restanti $m(m-1)$ lavori di durata 1 sulle altre $m-1$ macchine. Ognuna di queste macchine riceve m lavori, raggiungendo un carico finale di m.

  In questa configurazione ottimale, tutte le macchine terminano al tempo m. Il makespan della soluzione ottima è quindi $C^*_{\max} = m$.

Makespan: L'Euristica LPT

- L'algoritmo precedente è vulnerabile quando un lungo lavoro viene processato per ultimo.
- **Soluzione:** Ordinare i lavori dal più lungo al più corto.

Vediamo la variante detta LPT:

1. Ordina i lavori della lista per durata decrescente.
2. Per ogni lavoro della lista,
   - assegnalo alla macchina che al momento ha il carico di lavoro totale minore.

Makespan: Approssimazione $\frac{4}{3}$ per LPT (1/3)

Teorema: L'algoritmo LPT ha un rapporto di approssimazione $\rho \leq \frac{4}{3}$.

Dimostrazione:

Sia $p_1, p_2, \ldots p_n$ la sequenza dei lavori ordinata per durata decrescente, $C_{\max}$ il makespan prodotto dall'algoritmo e $C^*_{\max}$ il makespan ottimale.
Sia il k-mo lavoro quello che termina per ultimo nella schedulazione greedy e t_k l'istante in cui questo lavoro inizia la sua esecuzione.
Con lo stesso ragionamento usato nella prova della 2-approssimazione del greedy dove i lavori vengono assegnati in ordine qualunque si ottiene:

$$C_{\max} < C^*_{\max} + p_k.$$

Analizziamo i due possibili casi a seconda che la durata del lavoro p_k sia limitata o meno da $\frac{1}{3}C^*_{max}$. Dimostreremo che in entrambi i casi il rapporto non supera $\frac{4}{3}$.

Makespan: Approssimazione $\frac{4}{3}$ per LPT (2/3)

1. $p_k \leq \frac{1}{3} C^*_{max}$. In questo caso la dimostrazione è semplice:

$$C_{\max} < C^*_{\max} + p_k \leq C^*_{\max} + \frac{1}{3} C^*_{max} = \frac{4}{3} C^*_{max}.$$

Da cui ovviamente

$$\frac{C_{\max}}{C^*_{\max}} < \frac{4}{3}$$

2. $p_k > \frac{1}{3} C^*_{max}$. Nella prossima slide dimostreremo che l'algoritmo greedy in questo caso non sbaglia, ovvero $C_{\max} = C^*_{\max}$, da cui si ha che il rapporto è $1 < \frac{4}{3}$.

Makespan: Approssimazione $\frac{4}{3}$ per LPT (3/3)

- $p_k > \frac{1}{3} C^*_{max}$.
  Poiché tutti i lavori precedenti sono almeno lunghi quanto p_k, segue che:
  - ogni lavoro ha durata $> \frac{C^*_{\max}}{3}$
  - quindi su ogni macchina possono essere assegnati al massimo 2 lavori, altrimenti la somma supererebbe $C^*_{\max}$

  Poiché LPT cerca di bilanciare il carico, e nessuna macchina può ricevere più di due lavori, allora:
  - il numero minimo di macchine necessario è $\geq \lceil \frac{n}{2} \rceil$
  - LPT assegna esattamente un massimo di 2 lavori per macchina
  - tutte le macchine raggiungono un carico massimo pari a $C^*_{\max}$

  Conclusione: se l'ultimo lavoro ha durata $> \frac{C^*_{\max}}{3}$, allora **LPT produce una soluzione ottima**.

LPT: Implementazione in Python

- Un'implementazione diretta dell'algoritmo richiede, per ciascuno degli n lavori, di trovare la macchina con il carico minimo per poter effettuare l'assegnazione. Senza strutture dati specifiche, questa ricerca implica scorrere ogni volta la lista degli attuali carichi delle m macchine, un'operazione dal costo di $O(m)$. Di conseguenza, la complessità dell'intera fase di assegnazione è $O(nm)$.

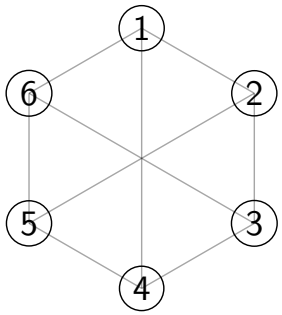
- Utilizzando un **min-heap**, si può ottenere la macchina che si libera prima (quella con il carico minimo) in tempo $O(\log m)$. Di conseguenza, la complessità dell'intera fase di assegnazione si riduce drasticamente a $O(n \log m)$, rendendo l'algoritmo molto più efficiente, specialmente quando il numero di macchine m è grande.

Python: LPT con min-heap ($O(nlogm)$)

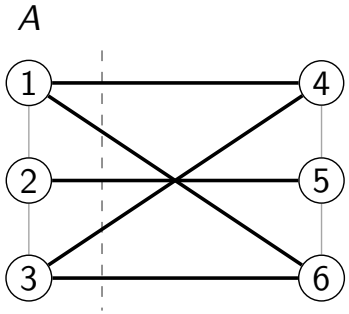
```python
import heapq

def lpt_schedule(durata_lavori, num_macchine):
    """
    Calcola il makespan usando l'algoritmo LPT con un min-heap
    Restituisce una tupla contenente:
        - Il makespan (un float).
        - Una lista con i carichi finali di ogni macchina.
    """
    # Ordina i lavori in ordine decrescente di durata.
    lavori_ordinati = sorted(durata_lavori, reverse=True)
    # Inizializza un min-heap per le macchine.
    # Ogni elemento è una tupla: (carico, indice_macchina).
    H = [(0, i) for i in range(num_macchine)]
    # Lista per memorizzare i carichi finali delle macchine
    carichi = [0] * num_macchine
    #  Assegna ogni lavoro usando l'heap.
    for durata in lavori_ordinati:
        # Estrai la macchina con il carico minimo
        carico_min, indice_macchina = heapq.heappop(H)
        # Calcola il nuovo carico e aggiorna la nostra lista di risultati
        nuovo_carico = carico_min + durata
        carichi[indice_macchina] = nuovo_carico
        # Reinserisci la macchina con il suo carico aggiornato nell'heap
        heapq.heappush(H, (nuovo_carico, indice_macchina))
    makespan = max(carichi)
    return makespan, carichi_finali
```

Problema MAX-CUT: Definizione

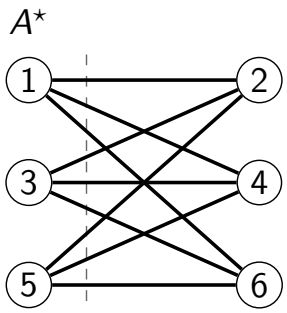
Dato un grafo G , un **taglio** di G è un sottoinsieme A dei suoi nodi. Il valore di un taglio è il numero di archi con un estremità in A e l'altra in $V \setminus A$. L'obiettivo è trovare un taglio di valore massimo.
Nella figura, a sinistra: un grafo G; al centro: un taglio qualsiasi; a destra: un taglio massimo (archi di attraversanto evidenziati)



Grafo originale



Un taglio di valore 5



Un taglio massimo

Greedy per MAX-CUT

Questo algoritmo greedy è semplice e si può implementare in tempo $O(m)$.

1. Parti con gli insiemi-partizioni A e B vuoti.
2. Assegna i nodi di G agli insiemi-partizioni in ordine di indice crescente:
   - Assegna il nodo i all'insieme-partizione che contiene meno suoi vicini.

Analisi della 2-approx per MAX-CUT

- **Teorema** L'algoritmo ha un rapporto di approssimazione $\rho \leq 2$.

Dimostrazione Sia k il valore prodotto dall'euristica e k^* il valore della soluzione ottima.
Per analizzare il contributo di ogni vertice, associamo in modo univoco ogni arco a uno dei suoi due estremi. Convenzionalmente, stabiliamo che ogni arco (u, v), con $u < v$, sia "di responsabilità" del vertice con indice maggiore, v (se $u < v$ l'arco (u, v) è responsabilità di v).
Sia r_i il numero di archi di cui il vertice i è responsabile (ovvero, il numero di suoi vicini con indice minore di i). Dimostriamo ora le seguenti due diseguaglianze

1. **Limite superiore della soluzione ottima**. Per costruzione, ogni arco viene contato esattamente una volta, quindi la somma di tutte le responsabilità è pari al numero totale di archi: $\sum_{i=1}^{n} r_i = m$. Il valore della soluzione ottima, k^*, non può evidentemente superare il numero m di archi del grafo. Si ha quindi

$$k^* \leq m = \sum_{i=1}^{n} r_i$$

2. **Limite inferiore per la soluzione greedy**. Quando l'algoritmo decide dove collocare il nodo i, considera solo gli r_i archi che lo collegano ai vertici con indice minore (già assegnati). La regola greedy consiste nel posizionare i nell'insieme che massimizza il numero di archi tagliati tra questi r_i. Il numero di nuovi archi tagliati è quindi almeno $\frac{r_i}{2}$.
Il valore totale del taglio greedy, k, è la somma dei contributi di tutti i vertici. Possiamo quindi affermare che: $k \geq \sum_{i=1}^{n} \frac{r_i}{2}$ ovvero

$$\sum_{i=1}^{n} r_i \leq 2 \cdot k$$

Mettendo insieme le due diseguaglianze otteniamo la tesi:

$$k^* \leq \sum_{i=1}^{n} r_i \leq 2 \cdot k$$

Da cui si ricava $\frac{k^*}{k} \leq 2$ il che dimostra che il rapporto d'approssimazione è limitato da 2.

MAX CUT: caso limite (1/2)

Consideriamo un'istanza del problema MAX CUT, definita su un grafo con n nodi costruito nel seguente modo:

> Due nodi, che chiameremo v_1 e v_2, sono ciascuno connessi a tutti gli altri (hanno quindi grado $n-1$).
>
> Ciascuno degli altri $n-2$ nodi è collegato solo a v_1 e v_2 (hanno quindi grado 2).

Per questa istanza dimostreremo che:

> **Il valore k della soluzione greedy** è: $k = n-1$.
>
> **Il valore k^* della soluzione ottima** è: $k^* = 2(n-2)$.

Il rapporto tra il valore della soluzione ottima e quello ottenuto dall'euristica greedy è:

$$\frac{k^*}{k} = \frac{2(n-2)}{n-1} = \frac{2(n-1)-2}{n-1} = 2 - \frac{2}{n-1}$$

Al crescere di n, la frazione $\frac{2}{n-1}$ tende a zero e il rapporto si avvicina a 2. Questo esempio mostra che esistono istanze per cui il rapporto di approssimazione dell'euristica greedy è arbitrariamente vicino a 2, dimostrando che il bound di approssimazione non può essere migliorato.

MAX CUT: caso limite (2/2)

Analizziamo in dettaglio il comportamento dell'euristica greedy su questa istanza:

- **Calcolo di k:** L'euristica greedy processa i nodi in ordine numerico crescente.
  1. Il nodo v_1 viene inserito nell'insieme A.
  2. Il nodo v_2, che ha come unico vicino già processato v_1 (che è in A), viene inserito in B.
  3. Gli altri nodi $(v_3, v_4, \ldots, v_n)$ sono ciascuno collegati sia a v_1 (in A) che a v_2 (in B).
     Qualsiasi sia l'insieme in cui vengono collocati, contribuiranno sempre con con il taglio di un solo arco.

  Dunque il valore del taglio prodotto dall'euristica greedy è:

$$k = 1 + (n - 2) = n - 1$$

- **Calcolo di k^*:** Se poniamo entrambi i nodi v_1 e v_2 nello stesso insieme A, e tutti gli altri nodi in B, ciascuno dei $n-2$ nodi in B vedrà tagliati entrambi i suoi archi (uno verso v_1, uno verso v_2).

  Il valore del taglio in questo caso sarà: $k^* = 2(n-2)$

Implementazione in Python

```
def EuristicaTaglioMassimo(G):
    """
    Restituisce la lista A coi nodi del taglio.
    """
    n = len(G)
    # IA[i]: "incentivo" per il vertice i ad andare in A.
    # Corrisponde al numero di vicini di 'i' (con indice < i)  già collocati in B.
    IA = [0] * n
    # IB[i]: "incentivo" per il vertice i ad andare in  B.
    # Corrisponde al numero di vicini di 'i' (con indice < i) già collocati in A.
    IB = [0] * n
    A = []
    for i in range(n):
        if IA[i] >= IB[i]:
            A.append(i)
            for u in G[i]:
                if u > i:
                    # u è un vicino non ancora processato
                    # Poiché 'i' è in A, aumento l'incentivo
                    # per il suo vicino 'u' ad andare in B
                    IB[u] += 1
        else:
            for u in G[i]:
                if u > i:
                    # u è un vicino non ancora processato
                    # Poiché 'i' è in B, aumento l'incentivo
                    # per  il suo vicino 'u' ad andare in A.
                    IA[u] += 1
    return A
```

- La complessità dell'algoritmo è $O(n+m)$

Bin Packing: definizione del problema

- Abbiamo n oggetti con dimensioni intere e una scorta illimitata di contenitori identici di capacità c. L'obiettivo è raggruppare gli oggetti in modo da minimizzare il numero di contenitori necessari.

Esempio: con $c = 10$ e oggetti $[6, 5, 4, 4, 3, 2, 2]$ si può impacchettare tutto in 3 contenitori:

- Contenitore 1: $[6, 4]$ (Uso: $10/10$)
- Contenitore 2: $[5, 3, 2]$ (Uso: $10/10$)
- Contenitore 3: $[4, 2]$ (Uso: $6/10$)

Bin Packing: euristica Next Fit

Non si conoscono algoritmi efficienti per la soluzione ottima. Perciò usiamo euristiche che in tempo polinomiale forniscono soluzioni accettabili.

Ecco di seguito l'euristica nota come **Next Fit**:

1. Inserisci il primo oggetto nel contenitore 1
2. Per ogni oggetto successivo, se entra nel contenitore corrente, inseriscilo; altrimenti, chiudi il vecchio contenitore, inaugura un nuovo contenitore e inserisci lì l'oggetto.

Next Fit: analisi del rapporto 2

- L'algoritmo ha un rapporto di approssimazione $\rho < 2$.

Dimostrazione Sia k il numero di scatole utilizzate dall'algoritmo greedy e k^* il numero di scatole in una soluzione ottima. Ed indichiamo con s la somma totale delle dimensioni di tutti gli n oggetti.
La dimostrazione si basa su due disuguaglianze principali.

1. **Limite Inferiore per la Somma delle Dimensioni** Ogni scatola (tranne l'ultima) è stata chiusa perché l'oggetto successivo non ci stava. Ciò significa che la somma del peso di quella scatola e il peso dell'oggetto che ha causato la sua chiusura supera la capacità c.
   Se considero la somma delle dimensioni tra due contenitori consecutivi questa dunque deve essere superiore a c. Se k è pari, abbiamo $k/2$ coppie quindi ho $c \cdot k/2 < s$. Se k è dispari devo escludere l'ultima scatola ed ho $(k-1)/2$ coppie quindi ho $c \cdot (k-1)/2 < s$. Per evitare di distinguere il caso pari e dispari possiamo considerare la somma dei primi $k-1$ contenitori che va bene per entrambi i casi. Abbiamo dunque:
   $$c\frac{k-1}{2} < s$$
2. **Limite Superiore per la Somma delle Dimensioni**. Il numero di contenitori in una soluzione ottima, k^*, deve essere sufficiente a contenere la somma totale delle dimensioni degli oggetti. Poiché ogni contenitore ha capacità c, si ha necessariamente:
   $$s \leq c \cdot k^*$$
   Questa è una stima per difetto del numero di contenitori necessari.

Ora combiniamo le due diseguaglianze sostituendo la prima della seconda:

$$c\frac{k-1}{2} < c \cdot k^*$$

Semplificando, otteniamo:

$$k - 1 < 2k^*$$

Poiché k e k^* sono numeri interi $k - 1 < 2k^*$ equivale a dire $k \leq 2k^*$. Questo dimostra che il rapporto di approssimazione dell'algoritmo è limitato da 2.

Next Fit: caso limite (1/2)

Consideriamo la seguente istanza:

- **Capacità dei contenitori**: c
- **Lista di oggetti**: $2c$ oggetti, presenti in una sequenza che alterna un oggetto di dimensioni $c/2$ e uno di dimensioni 1 (la lista è: $[c/2, 1, c/2, 1, .., c/2, 1]$ e contiene c oggetti di dimensione $c/2$ e c oggetti di dimensione 1).

Su quest'istanza proveremo che:

- **Il valore k della soluzione greedy** è $k = c$.
- **Il valore k^* della soluzione ottima:** è $k^* = c/2 + 1$

Il rapporto tra valore dell'euristica e quello della soluzione ottima è:

$$\frac{k}{k^*} = \frac{c}{c/2+1} = \frac{2c}{c+2} = \frac{2(c+2)}{c+2} - \frac{4}{c+2} = 2 - \frac{4}{c+2}$$

Al crescere di c, la frazione $\frac{4}{c+2}$ tende a zero e il rapporto tende a 2.
Questo esempio dimostra che esistono istanze per cui il rapporto di approssimazione è arbitrariamente vicino a 2, e quindi il bound sull'approssimazione dell'euristica greedy non è migliorabile.

Next Fit: caso limite (2/2)

Analizziamo in dettaglio cosa accade per quest'istanza:

- **il valore k della soluzione greedy:** I primi due oggetti di dimensione c ed 1 vengono inseriti nella prima scatola dove non resta spazio per il terzo oggetto di dimensione c. Questo pattern si ripete per c volte. Si ottiene quindi $k = c$.

- **il valore k^* della soluzione ottima:** La strategia ottima consiste nel raggruppare a coppie i c oggetti di dimensione $c/2$ utilizzando c scatole ed utilizzare una scatola ulteriore per i restanti c oggetti di dimensione 1. Si ottiene quindi $k^* = c + 1$.

Implementazione Next Fit in Python

```
def inscatolamento(L, c):
    """
    Implementa l'euristica greedy per l'inscatolamento.
    Restituisce una lista di liste che rappresenta
    il contenuto di ogni scatola, utilizzando
    gli indici  degli oggetti.
    """
    n = len(L)
    S = [[0]]
    peso_scatola = c-L[0]
    for i in range(1,n):
        if peso_scatola - L[i] >= 0:
            # l'oggetto entra nella scatola corrente
            S[-1].append(i)
            peso_scatola -= L[i]
         else:
            #Chiudiamo la scatola corrente, ne apriamo
            #  una nuova e vi inseriamo l'oggetto
            S.append([i])
            peso_scatola = c-L[i]
    return S
```

- La complessità dell'algoritmo è $O(n)$