1 Il problema del massimo taglio

Dato un grafo G non diretto e connesso un taglio di G è un sottoinsieme A dei suoi vertici. Il valore di un taglio è dato dal numero di archi che hanno esattamente un estremo in A. Vogliamo trovare il taglio di valore massimo.

Si propone il seguente algoritmo greedy:
parti con due insiemi A e B vuoti. Considera i vertici del grafo G nell'ordine crescente e inseriscili in A o B in base alla regola che segue:
Ogni nodo i può contribuire al taglio che si va via via creando apportando potenzialmente nuovi archi (quelli che incidono su di lui e sui nodi già considerati). Sia x il numero di questi archi. Inserisci i in A se così facendo il contributo al taglio è almeno $\frac{x}{2}$, inseriscilo in B altrimenti. Dopo aver inserito ogni nodo nell'insieme opportuno restituisci A.

In figura 1 viene riportata una possibile implementazione in python di compessità $O(m)$:

```
def MC(G):
    n = len(G)
    CA = [0]*n
    CB = [0]*n
    A=[]
    for i in range(n):
        if CA[i] >= CB[i]:
            A.append(i)
            for u in G[i]:
                if u>i: CB[u] += 1
        else:
            for u in G[i]:
                if u>i: CA[u] += 1
    return A
```

Figure 1: Euristica greedy per il problema del taglio massimo.

Teorema 1. *L'euristica ha un rapporto d'approssimazione limitato da* 2

Proof. Sia k il valore della soluzione greedy e k^* quello della soluzione ottima. Assegnamo la *responsabilità* di ciascun arco al suo estremo di indice massimo. Sia r_i il numero di archi di cui il nodo i è responsabile. Ovviamente vale

$$k^* \leq m = \sum_{i=0}^{n-1} r_i \tag{1}$$

Per la regola con cui viene assegnato il nodo i ad A o a B si ha che il contributo di i al taglio è almeno $\frac{r_i}{2}$. Quindi vale

$$\sum_{i=0}^{n-1} \frac{r_i}{2} \leq k \tag{2}$$

Dalle disequazioni 1) e 2) ricaviamo $k^* \leq \sum_{i=0}^{n-1} r_i = 2\sum_{i=0}^{n-1} \frac{r_i}{2} \leq 2k$ e quindi $\frac{k^*}{k} \leq 2$. □

Il rapporto d'approssimazione non può essere migliorato come mostra l' istanza con un grafo di n nodi con i nodi 0 e 1 di grado $n-1$ e tutti gli atri nodi di grado 2. L'euristica metterebbe il primo nodo in A ed il secondo in B a questo punto tutti i rimanenti nodi contribuirebbero al taglio per esattamente 1 (indipendentemente dall'insieme in cui vengono inseriti). Viene quindi prodotto un taglio di valore $n-1$. Mettendo i primi due nodi del grafo in A e tutti gli altri in B si otterrebbe un taglio di valore $2(n-2)$. Si ha quindi in questo caso

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

che tende a 2 al crescere di n.

Attualmente il miglior algoritmo approssimante per questo problema ha rapporto $\alpha \approx 1.389$.

2 Il problema dell'impacchettamento (BP)

Il problema dell'impacchettamento è un problema di ottimizzazione che consiste nell'impacchettare in scatole con capacità c un insieme di n oggetti ciascuno di dimensione intera al più c, minimizzando il numero di contenitori utilizzati.
Il problema richiede dunque di trovare la distribuzione degli oggetti all'interno dei contenitori in modo da usare il minor numero possibile di contenitori, rispettando le dimensioni dei singoli oggetti e dei contenitori stessi.
Il problema è notoriamente difficile e per ottenere soluzioni soddisfacenti in tempi ragionevoli si ricorre ad algoritmi di approssimazione. La più semplice euristica è di tipo greedy ed è nota come *Next Fit*, lavora come segue:

- Si considerano gli oggetti da impacchettare in sequenza. All'inizio abbiamo un unico contenitore contenente il primo oggetto. Per ogni altro oggetto, se il contenitore ha spazio per contenerlo allora l'oggetto viene inserito nel contenitore, in caso contrario il contenitore viene chiuso e viene inaugurato un nuovo contenitore.

In figura 2 si riporta un'implementazione in python di complessità $O(n)$.

La procedura prende in input la lista dei pesi degli oggetti e la capacità delle scatole e restituisce il numero di scatole necessarie. La complessità è $\Theta(n)$

```
def NF(P,c):
    n = len(P)
    Sol = [[0]]
    spazio = c - P[0]
    for i in range(1, n):
        if spazio - P[i] >= 0:
            Sol[-1].append(i)
            spazio -= P[i]
        else:
            Sol.append([i])
            spazio= c-P[i]
    return len(Sol), Sol

>>> P=[2, 5, 4, 7, 1, 3, 8], C=10
>>> NF(P,10)
(5, [[0, 1], [2], [3, 4], [5], [6]])
```

Figure 2: Euristica NF per il problema dell'impacchettamento.

Teorema 2. *L'euristica* Next_Fit *ha un rapporto d'approssimazione limitato da 2.*

Proof. Sia k il numero di contenitori utilizzato dall'euristica e k^* quello della soluzione ottima inoltre indichiamo con s la somma delle dimensioni degli n oggetti.
Deve ovvviamente aversi

$$s \leq c \cdot k^* \tag{3}$$

Nota che di due contenitori inaugurati in sequenza almeno uno è pieno per più della metà infatti se il primo dei due contenitori è pieno per meno della metà allora l'oggetto che ha portato all'inaugurazione del contenitore successivo deve avere una dimensione superiore alla metà di c. Questo significa che tutti i contenitori tranne l'ultimo sono pieni per più della metà. Abbiamo dunque

$$(k-1)\frac{c}{2} < s \tag{4}$$

Mettendo insieme le due diseguaglianze abbiamo $(k-1)\frac{c}{2} < c \cdot k^*$ da cui ricaviamo $k - 1 < 2k^*$ e, poichè k e k^* sono interi deve aversi $k \leq 2k^*$ da cui finalmente deduciamo $\frac{k}{k^*} \leq 2$ □

.

Il rapporto stabilito dal teorema non può essere migliorato come mostra la seguente istanza:
Considera una lista di $2c$ oggetti di peso c e $2c$ oggetti di peso 1 da dover inserire in scatole di dimensione $2c$.
La soluzione ottima consiste nell'inserire due oggetti di dimensione c in ciascuna

scatola e poi utilizzare due scatole per inserire i $2c$ oggetti di dimensione 1 Il numero minimo di scatole k^* necessarie è dunque $c+2$ scatole. Tuttavia se nella lista gli oggetti delle due diverse dimensioni si alternano l'algoritmo NF utilizzerà $2c$ scatole inserendo in ogni scatola un oggetto di dimensione c ed un oggetto di dimensione 1 producendo una soluzione che utilizza un numero di scatole k pari a $2c$. Per questa istanza si ha dunque rapporto

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

il rapporto al crescere di c tende a 2.

Nell'euristica precedente si considera un oggetto per volta senza conoscere il peso degli oggetti futuri (algoritmo online). Un problema con questi algoritmi è quello di avere oggetti pesanti tardi nella sequenza. Nell'euristica che segue si assume di avere di fronte tutti gli oggetti (algoritmo offline). L'idea dell'euristica FFD è quella di ordinare gli oggetti per peso decrescente e poi procedere con il loro impacchettamento con la seguente strategia: passa in rassegna le eventuali scatole già inaugurate, sistema l'oggetto nella prima che si incontra in grado di contenerlo, se nessuna scatola è in grado di farlo inaugura una nuova scatola. Le scatole vengono dunque sigillate solo quando tutti gli oggetti saranno stati inscatolati.

Considera scatole di dimensione $c = 10$ e lista di oggetti $lista = [5,\ 4,\ 3,\ 2,\ 2,\ 2,\ 2,\ 2]$ Gli oggetti possono essere partizionati in due scatole $(5, 3, 2)$ e $(4, 2, 2, 2)$ ma il metodo FFD richiede tre scatole poiché piazzerebbe gli oggetti di peso 5 e 4 insieme.

In figura 2 si riporta una possibile implementazione in python di complessità $O(n^2)$.

Teorema 3. *L'euristica* First_Fit_Decreasing *ha rapporto d'approssimazione* $\frac{3}{2} + \frac{1}{k^*}$.

Proof. Sia k il numero di contenitori utilizzato dall'euristica e k^* quello della soluzione ottima inoltre indichiamo con s la somma delle dimensioni degli n oggetti.
Dividiamo gli oggetti in 4 categorie:

- A=gli oggetti di dimensione $\left(\frac{2c}{3}, c\right]$
- B=gli oggetti di dimensione $\left(\frac{c}{2}, \frac{2c}{3}\right]$
- C=gli oggetti di dimensione $\left(\frac{c}{3}, \frac{c}{2}\right]$
- D=gli oggetti di dimensione $\left(0, \frac{c}{3}\right]$

```
def FFD(P,c):
    lista=[(P[i],i) for i in range(len(P))]
    lista.sort(reverse= True)
    Sol=[]
    C=[]
    for spazio, i in lista:
        s=0
        while s< len(C):
            if C[s]+ spazio <= c:
                C[s] += spazio
                Sol[s].append(i)
                break
            s+=1
        if s==len(C):
            Sol.append([i])
            C.append(spazio)
    return  len(Sol), Sol

>>> P=[2, 5, 4, 7, 1, 3, 8], c=10
>>> FFD(P,10)
(3, [[6, 0], [3, 5], [1, 2, 4]])
```

Figure 3: Euristica FFD per il problema dell'impacchettamento.

Cominciamo col far vedere che se nella soluzione greedy non compaiono oggetti di tipo D allora l'euristica non sbaglia e produce la soluzione ottima. Nota che gli oggetti di tipo A devono necessariamente comparire da soli in ciascuna scatola mentre in altre scatole possono esserci coppie di oggetti ma solo se i due oggetti sono uno di tipo B e l'altro di tipo C o entrambi di tipo C. Poiché gli oggetti sono ordinati per dimensione decrescente e l'euristica greedy li sistemerà nelle scatole considerandoli in quell'ordine, non nè difficile vedere che l'ottimo e la soluzione greedy utilizzeranno lo stesso numero di scatole.
Considera ora il caso in cui sono presenti oggetti di tipo D ma nella soluzione greedy non c'è alcuna scatola che contiene solo oggetti di questo tipo. In questo caso la presenza degli oggetti di tipo D non influenza il numero di scatole e quindi la soluzione prodotta è quella richiesta da tutti gli altri oggetti e sappiamo già che per quegli oggetti il greedy produce l'impacchettamento l'ottimo.

Resta da dimostrare cosa accade quando la soluzione greedy presenta almeno una scatola con soli oggetti di tipo D. L'ultima scatola utilizzata deve essere certamente di questo tipo ma questo significa che le scatole precedenti saranno piene tutte piene per almeno $\frac{2c}{3}$. Sia k il numero di contenitori utiliz-

zato dall'euristica e k^* quello della soluzione ottima inoltre indichiamo con s la somma delle dimensioni degli n oggetti. Abbiamo quindi

$$(k-1)\frac{2c}{3} < s \tag{5}$$

inoltre

$$s < c \cdot k^* \tag{6}$$

Mettendo insieme le due diseguaglianze otteniamo $(k-1)\frac{2}{3} < k^*$ da cui $\frac{k-1}{k^*} < \frac{3}{2}$ e infine $\frac{k}{k^*} < \frac{3}{2} + \frac{1}{k^*}$. □

Con una prova molto più complicata si dimostra che il rapporto è limitato da $\frac{11}{9} + \frac{4}{k^*}$.