Esercitazione 5: Greedy Algorithms

Giacomo Paesani

April 17, 2024

Esercizio 1. [16.1-2:3, [1]] Consideriamo il problema di trovare un sottoinsieme di cardinalità massima di intervalli disgiunti da un insieme qualsiasi di intervalli $\{a_i = [s_i, f_i]\}$. Un algoritmo greedy che produce una soluzione ottima consiste nel selezionare ad ogni passo un intervallo, tra quelli che sono disgiunti con quelli precedentemente scelti, che (0) ha l'estremo destro di valore minimo.

Consideriamo alternative idee per un algoritmo greedy per lo stresso problema. Selezionare ad ogni passo un intervallo, tra quelli che sono disgiunti con quelli precedentemente scelti, che:

(1) ha l'estremo sinistro di valore massimo.

(2) ha o l'estremo destro di valore minimo o l'estremo sinistro di valore massimo.

(3) ha ampiezza $f_i - s_i$ minima.

(4) interseca il numero minimo di intervalli rimanenti.

(5) ha l'estremo sinistro di valore minimo.

Per ogniuna di queste soluzioni proposte dimostrare che esse producono sempre soluzioni ottime o fornire un contro-esempio in cui la soluzione ottenuta non è ottima.

Soluzione 1. La strategia (1) è sostanzialmente equivalente alla strategia (0). Una maniera per vedere questo fatto è quella di cambiare segno a tutti gli intervalli, cioè invece di considerare il problema P con input $a_i = [s_i, f_i]$, consideriamo l'equivalente problema P' con input $b_i = [-f_i, -s_i]$. In breve,

dire che due problemi P e P' sono equivalenti significa che se so risolvere il problema P con una certa complessità allora so anche risolvere il problema P' *trasformando* la soluzione per P con un costo computazionale *basso*. Allora, adottare la strategia (0) al problema P', equivale ad adottare la strategia (1) al problema P. La strategia (2) è una strategia ibrida tra quella proposta nel testo dell'esercizio e la (1), quindi ogni soluzione ottenuta adottando la strategia (2) è ottima.

La strategia (3) non produce sempre soluzioni ottime e per affermare questo è sufficiente mostrare un esempio. Sia $a_1 = [1, 5]$, $a_2 = [4, 7]$ e $a_3 = [6, 12]$. Le ampiezze di a_1, a_2 e a_3 sono rispettivamente di 4, 3 e 6 quindi la strategia (3) produce come soluzione l'insieme $\{a_2\}$ mentre la soluzione ottima è $\{a_1, a_3\}$.

Per la strategia (4), il discorso è similare. Sia $a_1 = [1, 3]$, $a_2 = [2, 5]$, $a_3 = [4, 7]$, $a_4 = [6, 9]$ e $a_5 = [8, 10]$. Ogniuno degli intervalli a_2, a_3 e a_4 interseca altri due intervalli, in particolare a_2 interseca a_1 e a_3, a_3 interseca a_2 e a_4 e infine a_3 interseca a_2 e a_5. Se quindi al primo passo scelgo a_2 allora ottengo o la soluzione $\{a_2, a_4\}$ o $\{a_2, a_5\}$ ed entrambe hanno meno elementi della soluzione ottima $\{a_1, a_3, a_5\}$.

Anche la strategia (5) non produce sempre soluzioni ottime. Infatti, sia $a_1 = [1, 6]$, $a_2 = [2, 3]$ e $a_3 = [4, 5]$. Dato che l'intervallo a_1 è quello di estremo sinistro minimo, allora la soluzione ottenuta adottando la strategia (5) è $\{a_1\}$ quando invece quella ottima è $\{a_2, a_3\}$. □

Esercizio 2 (16.1-4, [1])**.** Sia A un insieme, una *partizione* di A è una collezione $\{A_i\}$ di sottoinsiemi di A tali che $A_i \cap A_j = \emptyset$ per ogni $i \neq j$ e $\bigcup A_i = A$. Fornire un algoritmo in pseudo-codice che, data una collezione di intervalli $A = \{a_i = [s_i, f_i]\}$, determina il numero minimo di sottoinsiemi che formano una partizione di A tali che in ogni sottoinsieme A_i gli intervalli sono due a due disgiunti.

Soluzione 2. Una strategia possibile sarebbe di usare l'algoritmo greedy descritto nel Esercizio 1 nella seguente maniera. L'insieme A_1 è un insieme di cardinalità massima di A in cui gli intervalli sono a due a due disgiunti. Più in generale si può pensare che l'insieme A_{k+1} è un sottoinsieme di cardinalità massima di $A \setminus (A_1 \cup \ldots \cup A_k)$ in cui gli intervalli sono due a due disgiunti. Questo procedimento continua finché l'insieme $A \setminus (A_1 \cup \ldots \cup A_k)$ è non vuoto.

Una soluzione alternativa è quella descritta nel Algoritmo 1. L'idea è di costruire contemporaneamente gli insiemi della partizione in maniera greedy: un intervallo con estremo destro di valore minimo tra quelli in cui non sono

stati messi ancora in alcun elemento della partizione viene messo, se esiste, nell'elemento della partizione con indice minimo in cui è disgiunto con tutti gli altri intervalli di quel elemento o un nuovo elemento della partizione viene creato.

Algorithm 1 Algoritmo per partizionare un insieme di intervalli in maniera che in ogni elemento gli intervalli siano compatibili.

Input: Un insieme di intervalli A, ordinati con estremo destro non-decrescente.
Output: numero minimo di elementi compatibili di una partizione di A.

```
 1: function IntervalColoring(A)
 2:     n = |A|
 3:     if n == 0 then
 4:         return 0
 5:     k = 1
 6:     for i = 1, ..., n do
 7:         t = 0
 8:         for j = 1, ..., k do
 9:             if isCompatible(A_j,a_i)==TRUE and t == 0 then
10:                 A_j.Add(a_i)
11:                 t = 1
12:         if t == 0 then
13:             k = k + 1
14:             A_k.Add(a_i)
15:     return k
16: function isCompatible(A,a*)
17:     for a ∈ a do
18:         if a ∩ a* ≠ ∅ then
19:             return FALSE
20:     return TRUE
```

Prima analiziamo la funzione **isCompatible**: essa prende in input un insieme di intervalli A e un intervallo a^* e controlla se a^* è disgiunto da ogni elemento di A. Nel caso questo non è vero, allora l'algoritmo ritorna `FALSE` (Linea 19) e ritorna `TRUE` altrimenti (Linea 20).

La funzione principale di questa soluzione è **IntervalColoring**: il nome deriva dal fatto che questo problema equivale al problema di colorare i vertici ristretta ad una classe di grafi chiamati *interval*. Se A è l'insieme vuoto, allora l'unica partizione di A è quella senza elementi: questo caso è coperto dal controllo **if** in Linea 3. Allora consideriamo il caso in cui A è non vuoto

e supponiamo che gli elementi di A sono ordinati in ordine crescente in base al valore dell'estremo destro. Con il ciclo **for** di Linea 8, cerchiamo se tra gli insiemi di intervalli già creati, $A_1, \ldots, A_k$, ne esiste uno che è compatibile con l'intervallo a_i che vorrei inserire: se tale insieme esiste, allora aggiungo a_i solo all'insieme con cui è compatibile di indice minimo. Altrimenti, cioè se a_i non è compatibile con nessun insieme precedentemente creato, necessariamente dovremmo creare un nuovo insieme A_{k+1} dove poter inserire a_j (controllo **if** in Linea 12). La variabile t e i controlli su essa servono ad assicurarsi che ogni intervallo di A viene inserito in esattamente un elemento della partizione. □

Definizione. Sia $G = (V, E)$ e $A \subseteq E$ un insieme di archi contenuto in un albero di copertura di peso minimo di G. Un arco e è *sicuro* per A se $A \cup \{e\}$ è ancora contenuto in un albero di copertura di peso minimo di G.

Teorema (Teorema dell'arco leggero)**.** Sia $G = (V, E)$ e $A \subseteq E$ un insieme di archi contenuto in un albero di copertura di peso minimo di G e sia $(S, V - S)$ un qualsiasi taglio di G che rispetta A. Allora un arco leggero che attraversa il taglio è sicuro per A.

Esercizio 3 (23.1-1, [1])**.** Sia $G = (V, E)$ un grafo non diretto e connesso con gli archi pesati. Dimostrare o confutare con un contro-esempio la seguente affermazione: se (u, v) è un arco di peso minimo, allora esiste un albero di copertura di costo minimo di G che contiene l'arco (u, v).

Soluzione 3. L'affermazione è vera e il resto della soluzione consiste in una sua dimostrazione. Costruiamo prima di tutto un albero minimo di copertura T di G usando uno degli algoritmi noti (come quello di Kruskal or di Prim). Se T contiene (u, v) come arco allora l'affermazione è dimostrata. Altrimenti, cioè se (u, v) non è un arco di T, esiste in T un cammino P da u a v. Concatenando P con l'arco (u, v) si ottiene un ciclo C di G. Vogliamo mostrare che l'albero di copertura $T' = (T \setminus (s, t)) \cup (u, v)$, per un qualsiasi arco $(s, t) \neq (u, v)$ di C, è di peso minimo. Supponiamo che esiste un arco (s, t) di C che ha peso strettamente maggiore di quello di (u, v), cioè $w(s, t) > w(u, v)$, allora T' è un albero di copertura di G con peso strettamente minore di T, che è una contraddizione. Allora ogni arco di C ha lo stesso peso $w(u, v)$, e quindi T' ha lo stesso peso di T e quindi è l'albero di copertura cercato.

Una altra possibile soluzione a questo esercizio può essere facilmente ottenuta usando il Teorema dell'arco leggero nella seguente maniera. L'insieme

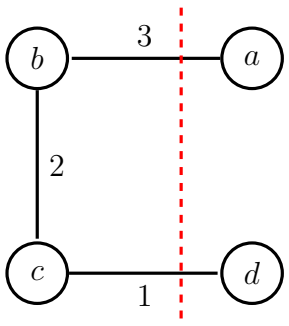
$A = \emptyset$ è ovviamente contenuto in un albero di copertura di costo minimo e il taglio $(\{u\}, V \setminus \{u\})$ rispetta A (come ogni altro taglio di G). Dato che (u, v) è un arco di peso minimo in G, è anche un arco leggero che attraversa $(\{u\}, V \setminus \{u\})$ e quindi è sicuro. Allora per definizione di sicuro e per costruzione, l'insieme $A \cup \{(u, v)\} = \{u, v\}$ è contenuto in un albero di copertura di peso minimo di G. □

Esercizio 4 (23.1-2:3, [1])**.** Sia $G = (V, E)$ un grafo, dimostrare o confutare con un contro-esempio le seguenti affermazioni.

(1) Sia $A \subseteq E$ un insieme di archi contenuto in un albero di copertura di peso minimo di G e sia $(S, V-S)$ un qualsiasi taglio di G che rispetta A. Un arco sicuro per A che attraversa il taglio è necessariamente leggero.

(2) Sia (u, v) un arco che appartiene a qualche albero di copertura di peso minimo di G. Allora (u, v) è leggero per un qualche taglio $(S, V - S)$ di G.

Soluzione 4. Un contro-esempio all'affermazione (1) è dato dal grafo in figura. Consideriamo il taglio $(\{b, c\}, \{a, d\})$ che rispetta l'insieme di archi $A = \{(b, c)\}$, allora si vede facilmente che l'arco (a, b) è sicuro ma non è leggero.



Dimostriamo che l'affermazione (2) è vera. Sia T un albero di copertura di peso minimo di G che contiene l'arco (u, v) e da esso rimuoviamo proprio (u, v), Allora si creano due alberi T_u e T_v che formano un taglio di G. Allora (u, v) è leggero per il taglio (T_u, T_v), infatti se non lo fosse e quindi esistesse un

arco (x, y) che attraversa il taglio e ha peso minore di (u, v) allora $T' = (T \setminus \{(u, v)\}) \cup \{(x, y)\}$ sarebbe un albero di copertura di G di peso strettamente inferiore a T, assurdo. □

Esercizio 5 (I. Salvo)**.** Sia $G = (V, E)$ un grado non diretto, connesso e con gli archi pesati con pesi tutti diversi e positivi. Sia T un albero di copertura di peso minimo di G, s un vertice di G e T_s l'albero dei cammini minimi da s verso tutti gli altri nodi di G. Dimostrare o confutare con un contro-esempio che T_s e T condividono almeno un arco.

Soluzione 5. L'affermazione che T_s e T condividono almeno un arco è vera. Sia (s, u) l'arco di costo minimo tra quelli incidenti ad s, allora necessariamente (s, u) è un arco di T_s per la disuguaglianza triangolare. Ci resta da dimostrare che (s, u) è anche un arco di T. Applicando l'algoritmo di Prim iniziando dal vertice s, per costruzione, viene selezionato (s, u) come arco di T. Dato che gli archi di G hanno pesi tutti diversi, l'albero minimo di copertura T di G è unico e quindi abbiamo mostrato che (s, u) è un arco comune agli alberi T_s e T. □

Esercizio 6 (23.1-11, [1])**.** Sia $G = (V, E)$ un grafo non diretto e connesso con gli archi pesati e T un albero di copertura di G di peso minimo. Supponiamo che il peso di uno degli archi non appartenenti a T diminuisce. Fornire un algoritmo in pseudo-codice che dato G, T e come viene modificato il peso dei un arco, trova un albero di copertura di peso minimo nel grafo modificato (senza calcolarlo da capo).

Soluzione 6. Sia (u, v) l'arco di G a cui viene cambiato peso da $w(u, v)$ in $w'(u, v)$. Dato che (u, v) non appartiene a T, allora esiste un cammino P in T che collega u e v. La soluzione consiste nel capire se è c'è un arco (s, t) in P tale che $w(s, t) > w'(u, v)$: se tale arco esiste allora è possibile trovare un nuovo albero di copertura di peso minimo. Altrimenti T resta minimo.

La soluzione proposta è quella data dal Algoritmo 2. Come prima cosa si esegue una ricerca in ampiezza BFS nell'albero T a partire da uno degli estremi dell'arco (u, v), ad esempio u: questa chiamata (Linea 2) restituisce il vettore dei padri. Tale vettore ci permette di ricostruire l'unico cammino P in T tra u e v. A questo punto cerchiamo tra gli archi di P, grazie al ciclo **while** in Linea 7, quello di peso massimo: gli estremi e il peso di questo arco è salvato nelle variabili s, t e m. L'algoritmo termina sostituendo in T l'arco (s, t) con (u, v). □

Algorithm 2 Algoritmo per aggiornare l'albero di copertura di costo minimo in seguito ad una riduzione di costo di un arco.

Input: $G = (V, E)$ grafo non diretto, connesso e con archi pesati, $T \subseteq E$, $(u, v) \in E$ e valore w'
Output: $T' \subseteq E$ albero di copertura di peso minimo

```
1: function AdaptMST(G,T,(u,v),w')
2:     Parent=BFS(T,u)
3:     m = w'
4:     s = u
5:     t = v
6:     z = v
7:     while Parent[z] ≠ z do
8:         if w(z, Parent[z]) > m then
9:             m = w(z, Parent[z])
10:            s = z
11:            t = Parent[z]
12:        z = Parent[z]
13:    T.Remove(s, t)
14:    T.Add(u, v)
15:    return T
```

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms. 2022.