ASD - Esercitazione 04/05/2017
Algoritmi Greedy

Tor VerLouvre Nel museo Tor VerLouvre c'‘e un lungo corridoio rettilineo in cui sono esposti n quadri nelle posizioni $0 \leq q_1 < q_2 < \cdots < q_n$. dove $\forall i, q_i \in \mathbb{R}^+$ Il direttore del museo deve decidere quanti custodi assumere per poter sorvegliare tutti i quadri, sapendo che ogni custode deve avere una postazione fissa e che da quella posizione riesce a sorvegliare tutti i quadri che si trovano a distanza al più d.

- Descrivere un algoritmo che prende in input un vettore ordinato $Q = \{q_1, \ldots, q_n\}$ e d, e in tempo $O(n)$ restituisce le posizioni dei custodi $c_1, \ldots, c_k$ in modo che tutti i quadri siano sorvegliati e il numero k di custodi sia il più piccolo possibile.
- Il direttore del museo vorrebbe risparmiare qualche stipendio e assumere ancora meno custodi. Spiegategli in modo rigoroso perchè non è possibile sorvegliare tutti i quadri con meno custodi di quanti ne usa il vostro algoritmo.

Soluzione Al fine di rappresentare il problema descritto nella traccia, possiamo modellare il corridoio con una semiretta ed immaginare ogni quadro come un punto su tale semiretta. Anche le soluzioni del problema, cioè l'insieme delle posizioni dei custodi, sono ulteriori punti sulla semiretta.

Una volta posizionato un generico custode in posizione c_j avremo che tutti i quadri posizionati nell' intervallo $[c_j - d, c_j + d]$ saranno coperti. Si vuole trovare l'insieme di punti $c_1, \cdots c_k$ (di cardinalità minima) che copra tutti i punti $q_1, \cdots, q_n$ presenti sulla semiretta. In Fig 1 è riportato un esempio di instanza e di soluzione ottima.

Si vuole progettare un algoritmo greedy che trovi l'insieme di posizioni da assegnare ai custodi. Seguendo un approccio greedy vogliamo quindi trovare un criterio localmente ottimo con cui scegliere la posizione c_1 disinteressandoci delle successive posizioni da scegliere, poi determinare c_2 ancora ignorando le scelte future, e così via fino a quando tutti i quadri sono stati coperti. Ma quale criterio porta ad una soluzione ottima?

Proviamo ora a dare un primo criterio: considerare i quadri da sinistra a destra e piazzare il custode davanti (sopra) al primo quadro non coperto. Inizialmente nessun custode è stato posizionato e quindi banalmente nessun quadro è stato coperto, allora l'algoritmo piazzerà il primo custode davanti al primo quadro ($c_1 = q_1$). Potenzialmente il primo custode potrebbe coprire più di un quadro, quindi l'algoritmo al fine di posizionare il secondo custode individuerà il primo quadro ancora scoperto e posizionerà il custode davanti ad esso. Si itera fino a che ci sono quadri da coprire.

Tale regola non porta sempre ad una soluzione ottima, infatti è semplice trovare un'istanza del problema sulla quale è facile verificare che la soluzione

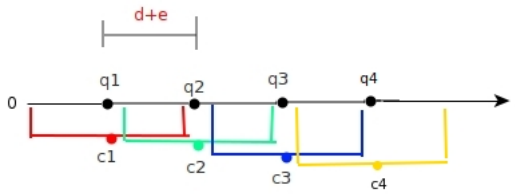

Figure 1: Esempio di posizionamento sub-ottimo dei custodi

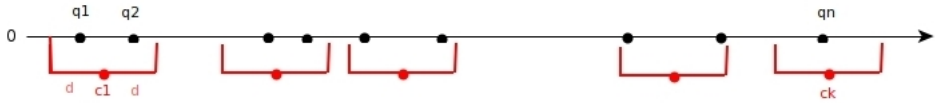

Figure 2: Esempio di posizionamento ottimo dei custodi

restituita dall'algoritmo non è ottima. Nell'istanza proposta ogni quadro (eccetto il primo) si trova a distanza $d + \epsilon$ (dove ϵ è una qualsiasi piccola costante positiva) dal precedente, come visibile in Figura 1 viene posizionato un custode per ogni quadro. Si può notare che semplicemente spostando il primo custode dalla posizione c_1 a $c_1 + \epsilon$, questo coprirebbe non più soltanto il primo quadro ma anche il secondo rendendo così superflo il custode posizionato in c_2. Cosa c'è che non va nel criterio proposto? Nella figura si può facilmente notare che il primo custode va a coprire inutilmente una porzione di semiretta a sinistra del punto q_1, mentre tra un quadro e l'altro c'è una porzione di semiretta coperta da 2 custodi, evidentemente risulta essere uno "spreco" far controllare la stessa porzione di corridoio a più di un custode. Queste semplici osservazioni ci suggeriscono il corretto modo in cui effettuare le scelte locali: ad ogni iterazione **si dovrebbe posizionare i custodi più a destra possibile non lasciando quadri scoperti alla loro sinistra**. Qualitativamente, applicando tale criterio avremo che il campo visivo di un generico custode terminando più a destra possibile minimizzerà la porzione di corridoio ancora da controllare. Come posso ottenere tale proprietà? Mi basta posizionare il primo custode in modo che il suo campo visivo inizi esattamente dove è posizionato il primo quadro , quindi $c_1 = q_1 + d$ coprendo così tutti i quadri $q_i : 0 \leq q_i \leq q_i + 2d$. Per posizionare il secondo custode si andrà a cercare il primo quadro ancora scoperto e sia q_j la sua posizione andremo a posizionare il custode in modo che il suo campo visivo inizi in q_j, quindi $c_2 = q_j + d$, saranno così coperti tutti i quadri $q_i : 0 \leq q_i \leq q_j + 2d$. Si procede seguendo questa regola fino a quando tutti i quadri sono stati coperti. In Figura 3 è mostrato il risultato ottenuto applicando la regola descritta ad un'istanza del problema.

Ecco lo pseudo codice che implementa il criterio di scelta locale appena proposto.

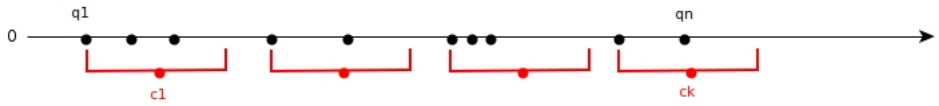

Figure 3: Esempio di posizionamento dei custodi del nostro algoritmo greedy

Algorithm 1: *greedyAlg*

Data: Q vettore ordinato contenente $q_1, \cdots, q_n$
Result: $C = \{c_1, \cdots, c_k\}$

```
1  begin
2      C = ∅
3      i = 1
4      last_position_covered = −1 // indica l'ultima posizione coperta
         dall'ultimo guardiano posizionato
5      for j = 1 to n do
6          if Q[j] > last_position_covered// il quadro non è stato ancora
             coperto then
7              c_i = Q[j] + d
8              last_position_covered = Q[j] + 2d
9              C = C ∪ {c_i}
10             i = i + 1
11     return C
```

L'algoritmo efffettua una semplice scansione del vettore dove per ogni elemento viene applicato un controllo effettuabile in tempo costante, da ciò deriva che la complessità dell'algoritmo è $O(n)$.

Dimostrazione dell'ottimalità dell'algoritmo.

Definiamo $O = \{o_1, \cdots, o_m\}$ come l'insieme di posizioni calcolate da un qualche algoritmo ottimo e $C = \{c_1, \cdots c_k\}$ l'insieme trovato dal nostro algoritmo (notare che, poichè O è definito ottimo, allora $m \leq k$). Non possiamo mostrare che $C = O$ (in quanto esistono istanze in cui diverse assegnazioni di posizioni coprono tutti i quadri usando lo stesso numero dei custodi), ma ci basta mostrare che $|C| = |O|$. Nella nostra dimostrazione useremo la tecnica nota come "greedy stays ahead".

L'idea è quella di comparare le soluzioni parziali dell'algoritmo greedy con le soluzioni parziali dell'algoritmo ottimo e mostrare che passo passo la soluzione dell'algoritmo greedy "sta avanti" rispetto alla soluzione ottima secondo una qualche proprietà definita ad hoc.

Siano $c_1, \cdots c_k$ le posizioni restituite dal nostro algoritmo ordinate in modo crescente e siano $o_1 \cdots o_m$ le posizioni in O anch'esse ordinate in modo crescente (ordinare la soluzione ottima secondo il criterio della soluzione greedy è un trucco che viene utilizzato spesso), vogliamo mostrare che l'r-esimo custode posizionato dal nostro algoritmo greedy non più a sinistra del r-esimo custode della soluzione ottima (i.e. $c_r \geq o_r$, in questo senso, "greedy stays ahead").

Lemma 1. $\forall r \leq m,\ c_r \geq o_r$

Proof. Per induzione su r:

caso base $r = 1$ Per costruzione

suppongo vero fino ad $r - 1$ **e dimostro per** r

Per ipotesi induttiva abbiamo che $c_{r-1} \geq o_{r-1}$. Sia q_h il primo quadro non coperto da $c_1, \ldots, c_{r-1}$ allora $q_h - c_{r-1} > d$ e applicando l'ipotesi induttiva ottiamo che $q_h - o_{r-1} \geq q_h - c_{r-1} > d$ quindi anche l'ottimo deve ancora coprire q_h. Il nostro algoritmo greedy piazzerà il prossimo custode in posizione $c_r = q_h + d$. Assumiamo per assurdo che $o_r > q_h + d$ allora neppure utilizzando il custode numero r il quadro q_h viene coperto, e poichè stiamo considerando i custodi ordinati da sinistra a destra neppure i prossimi potranno coprire q_h ($\forall j > r, o_j > o_r > q_h + d$). Quindi l'ottimo non coprirebbe il quadro q_h il che è assurdo. □

Dopo aver mostrato che passo passo il nostro algoritmo posiziona il prossimo custode non più a sinistra della posizione scelta da un algoritmo ottimo utilizziamo questa proprietà per dimostrare per assurdo che anche il nostro algoritmo è ottimo.

Teorema 1. *La soluzione restituita dall'algoritmo 1 è ottima*

Proof. Supponiamo per assurdo che la soluzione C non sia di dimensione ottima, ovvero $m < k$ (Si ricordi $m = |O|, k = |C|$ e che poichè O è una soluzione ottima allora $m \leq k$).

Per l'ipotesi m è l'ultimo custode della soluzione ottima (quindi non esistono ulteriori quadri da coprire), e poichè $m < k$ il nostro algorimo deve coprire almeno ancora un quadro q_h (i.e. $q_h - c_m > d$). Applicando il lemma precedente per $r = m$ possiamo dire che $c_m \geq o_m$ quindi abbiamo che $q_h - o_m \geq q_h - c_m > d$. Questo significa che l'ottimo deve ancora coprire il quadro q_h, il che è assurdo. □

The Project Manager Subito dopo esserti laureato con il massimo di voti in Informatica hai ottenuto un ambitissimo posto come Project Manager. Il tuo primo giorno di lavoro hai scoperto che ci sono $P = \{1, 2, \ldots, n\}$ progetti software da terminare e hai una sola squadra di informatici a cui assegnarli. I tuoi analisti ti informano che per ogni $i \in P$ per terminare quel progetto è necessario un tempo $t_i \in \mathbb{R}$ e che inoltre alcuni clienti sono più esigenti di altri. Infatti, per ogni $i \in P$, definito C_i come il momento in cui viene terminato il progetto i, allora il cliente di quel progetto sarà insoddisfatto in misura $w_i \cdot C_i$ (con $w_i \in \mathbb{R}$). Determinare un ordine in cui assegnare i progetti alla tua unica squadra in modo da minimizzare la somma delle insoddisfazioni dei clienti $\sum_{i \in P} w_i \cdot C_i$. Convincere rigorosamente il Consiglio di Amministrazione che non è possibile fare di meglio.

Soluzione Intuitivamente bisognerebbe soddisfare prima i clienti più esigenti con progetti più corti. E' facile trovare controesempi in cui qualsiasi algoritmo greedy che considera solamente i valori t_i (crescenti) o solamente i valori w_i (decrescenti) non trova l'ottimo. L'algoritmo greedy ottimo proposto consiste nell'ordinare i progetti per valori $\frac{w_i}{t_i}$ (informalmente una specie di insoddisfazione al secondo) decrescenti e scegliere i progetti in quell'ordine.

Dimostrazione dell'ottimalità dell'algoritmo.

Sia $S = (t_1, w_1), (t_2, w_2), \ldots, (t_n, w_n)$ una qualsiasi soluzione ordinata secondo un qualsiasi criterio. Definiamo "inversione" tra due progetti i e j se nella soluzione S $\frac{w_i}{t_i} < \frac{w_j}{t_j}$.

Il costo della soluzione di S è $c(S) = \sum_i C_i \cdot w_i$. Notare che il tempo C_i in cui un qualsiasi progetto i viene completato è definito dalla somma di tutti i tempi di completamento t_j dei progetti j scelti prima di i (i.e. $C_i = \sum_{j<i} t_j$) quindi

$$c(S) = \sum_i C_i w_i = \sum_i w_i \sum_{j \leq i} t_j = w_1\, t_1 + w_2(t_1 + t_2) + w_3(t_1 + t_2 + t_3) + \ldots w_n(t_1 + \cdots + t_n)$$

Se esiste una inversione allora esiste una coppia di progetti adiacenti i e $i+1$ tali che vale $\frac{w_i}{t_i} < \frac{w_{i+1}}{t_{i+1}}$ che moltiplicando a croce diventa

$$w_i t_{i+1} < t_i w_{i+1} \tag{1}$$

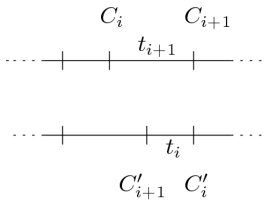

Figure 4: Quantità in gioco durante l'inversione del progetto i e del progetto $i+1$

Consideriamo il costo della nuova soluzione S' ottenuta invertendo il progetto i con il progetto $i+1$ e chiamiamo C'_j il tempo di completamento di un qualsiasi progetto in posizione j-esima nella nuova soluzione.

E' immediato verificare che invertire due progetti non cambia i tempi di completamento dei progetti precedenti e quelli successivi. Inoltre il tempo di completamento del progetto i è lo stesso di $i+1$ prima dell'inversione (i.e. $C'_i = C_{i+1}$), mentre $C'i+1 = C_{i+1} - t_i$ (si faccia riferimento alla Figura 4) Da queste osservazioni segue

$$\begin{aligned} c(S) - c(S') &= w_i C_i + w_{i+1} C_{i+1} - w_i C'_i - w_{i+1} C'_{i+1} \\ &= w_i C_i + w_{i+1} C_{i+1} - w_i C_{i+1} - w_{i+1}(C_{i+1} - t_i) \\ &= w_i (C_i - C_{i+1}) - w_{i+1}(C_{i+1} - (C_{i+1} - t_i)) \\ &= w_i t_{i+1} - w_{i+1} t_i \qquad (2) \\ &> 0 \end{aligned}$$

dove nell'ultima disequazione abbiamo utilizzato la disequazione (1). Abbiamo dunque mostrato che effettuando una inversione non peggioriamo la soluzione di S. Sia allora O una qualsiasi soluzione ottima con delle inversioni, risolviamo le inversioni (che sono un numero finito) fino ad ottenere una nuova soluzione O^* non peggiore di O (e quindi ancora ottima) senza inversioni. La soluzione restituita dal nostro algoritmo è senza inversioni. Rimane solo da mostrare che tutte le soluzioni senza inversioni hanno lo stesso costo.

Questo è banalmente vero perché le uniche differenze tra due soluzioni senza inversioni è l'ordinamento scelto tra due progetti i e j tali che $\frac{w_i}{t_i} = \frac{w_j}{t_j}$ (cioè $w_i t_j = w_j t_i$). Utilizzando questa proprietà a partire dall'equazione (2) si ottiene la tesi. □