Algoritmi di approssimazione

A.A. 2017-2018

Rapporto di approssimazione
Algoritmi «greedy»

Soluzioni ottime vs soluzioni approssimate

Spesso nelle applicazioni ci si trova di fronte alla necessita` di risolvere problemi NP-completi per i quali garantire l'ottimalita` della soluzione richiederebbe un tempo di calcolo troppo elevato. Ad esempio, sarebbe inaccettabile impiegare quasi mezza giornata per risolvere il problema dello scheduling dei lavori giornalieri in una fabbrica.
In molti casi è possibile "accontentarsi" di una soluzione sub-ottima, calcolabile in tempo polinomiale, cioè di una soluzione "ragionevole", pur non essendo la migliore **teoricamente** possibile.

Per valutare l'accettabilità di una soluzione approssimata, ci si basa sulle differenze tra soluzione approssimata e soluzione ottima esprimibile con il rapporto limite o di approssimazione

Rapporto limite

Per ogni istanza I di un problema di *ottimizzazione P* sia *S(I)* l'insieme delle soluzioni ammissibili per I;

- Ad ogni soluzione ammissibile S dell'istanza I, si associa una misura, ***costo*** di S: C(I,S);

- Se *P* è un problema di minimizzazione una soluzione ottima per l'istanza I sarà una soluzione S*:

$$C(I,S^*) = \min_{S \in S(I)} C(I,S)$$

- Se *P* è un problema di massimizzazione una soluzione ottima per l'istanza I sarà una soluzione S*:

$$C(I,S^*) = \max_{S \in S(I)} C(I,S)$$

Rapporto limite

Un algoritmo approssimato ha rapporto limite **ρ(n)** se per ogni istanza I di dimensione n il costo C(I, S) della soluzione S prodotta dall'algoritmo approssimato si discosta al più di un fattore **ρ(n)** dal costo C(I,S*) di una soluzione ottima:

$$\max \left[\frac{C(I,S)}{C(I,S^*)} , \frac{C(I,S^*)}{C(I,S)} \right] <= \rho(n)$$

Tale rapporto è:

- uguale a 1 per gli algoritmi ottimi
- sempre maggiore di 1 per gli algoritmi approssimati
- non negativo, supponendo la funzione costo sempre positiva

Quanto maggiore è il **rapporto limite** tanto peggiore sarà la soluzione approssimata rispetto a quella ottima.

Rapporto limite

Per dimostrare che una soluzione ammissibile ha un fattore di approssimazione “garantito”, dobbiamo confrontarla con la soluzione ottima, computazionalmente difficile da trovare.
Nell’analisi degli algoritmi approssimati si deve fare i conti con questa difficolta`.

Per alcuni problemi sono stati sviluppati algoritmi approssimati che hanno un *rapporto limite indipendente* dalla dimensione dei dati in input.

Per altri problemi esistono algoritmi approssimati che possono raggiungere *rapporto limite sempre più piccolo*, a spese di un tempo di esecuzione sempre maggiore.

Questi algoritmi ricevono in input non solo un’istanza del problema ma anche un valore $\varepsilon > 0$, tale che per ogni ε prefissato, l’algoritmo risulta essere $(1 + \varepsilon)$-approssimato.
Essi sono detti **schemi di approssimazione**.

4 tecniche generali

1. Algoritmi greedy: algoritmi semplici e veloci, ma si deve trovare una regola greedy che porti a soluzioni che si possono dimostrare «vicine» alla soluzione ottima
2. Metodo pricing: nome motivato da una prospettiva economica. Si considera un prezzo da pagare per rafforzare ogni vincolo del problema.
3. Programmazione lineare: si esplorano le relazioni tra la fattibilita` computazionale della programmazione lineare e la potenza espressiva della «piu` difficile» programmazione intera.
4. Programmazione dinamica: usando la programmazione dinamica su una versione arrotondata dell'input si possono ottenere approssimazioni molto buone.

Algoritmi di approssimazione greedy: load balancing

Problema: nella versione di ottimizzazione il problema load balancing è il seguente:

Dati un insieme finito $P = \{p_1, p_2, \ldots, p_n\}$ di programmi tali che ogni programma p_i richiede un tempo intero di esecuzione t_i, e m processori identici, eseguire tutti i programmi sugli m processori nel più breve tempo possibile.

Si suppone di non avere vincoli di precedenza o di finestra temporale e si considera uno scheduling non preemptive (l'esecuzione dei programmi non può essere interrotta).

Indicando con T_i il tempo richiesto dalla macchina M_i per eseguire i lavori assegnati:

$$T_i = \sum t_j , j \in A_i$$

dove A_i sono i lavori assegnati alla macchina i-esima.

Si vogliono assegnare i lavori in modo da minimizzare il massimo carico su ogni macchina, cioe` il *makespan*.

Algoritmi di approssimazione greedy: load balancing

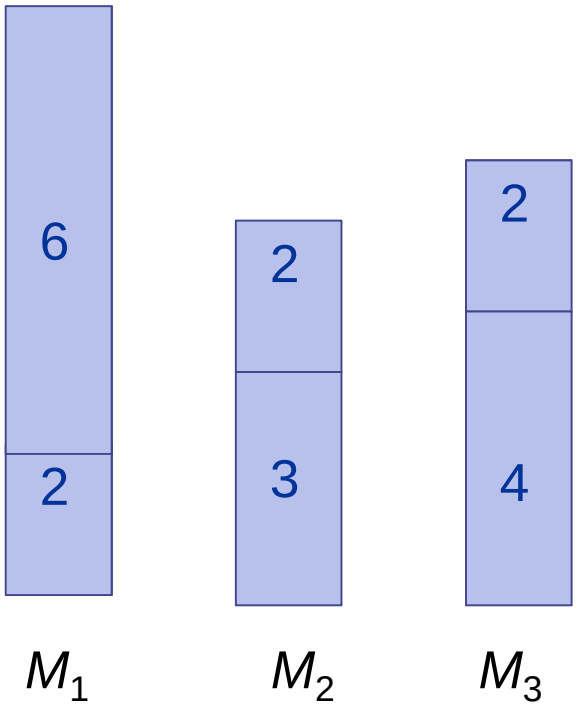
Consideriamo un algoritmo greedy molto semplice: si considerano i lavori *in un ordine qualunque* e li si assegna alla macchina piu` scarica nel momento in cui ognuno viene preso in considerazione.

```
Greedy-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = ∅ for all machines M_i
For j = 1, ..., n
  Let M_i be a machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

Complessita`: il primo ciclo for costa O(m), il secondo O(n.f(m)), dove f(m) è il costo della determinazione del minimo. Il secondo termine è dunque quello che determina la complessità .

Algoritmi di approssimazione greedy: load balancing

Nell'esempio è mostrata una sequenza di 6 attività con dimensione 2,3,4,6,2,2. Il makespan ottenuto è 8.



La soluzione trovata non è quella ottima; se l'ordine fosse stato 6,4,3,2,2,2, il makespan sarebbe stato 7 invece che 8!

Analisi del rapporto di approssimazione

Chiamiamo T il makespan risultante dall'assegnazione fatta dall'algoritmo; vogliamo dimostrare che T non è molto più grande del minimo makespan possibile T*, in particolare vogliamo dimostrare che il rapporto di approssimazione è 2.

T* non è noto e non sappiamo come calcolarlo

Idea: cerchiamo un limite inferiore per la soluzione ottima, ad esempio considerando il fatto che ognuna delle m macchine dovrà svolgere almeno 1/m del lavoro totale:

$$T^* \geq \frac{1}{m} \sum t_j$$

Analisi del rapporto di approssimazione

Questo lower bound non basta a dimostrare che il rapporto di approssimazione è 2: vi è un caso in cui questo limite è troppo debole.

Supponiamo che ci sia un'attività con un tempo di esecuzione molto maggiore rispetto al tempo totale di esecuzione delle altre. La soluzione ottima potrebbe consistere nel dedicare una macchina all'esecuzione di tale lavoro ed essa potrebbe essere l'ultima a terminare l'esecuzione.

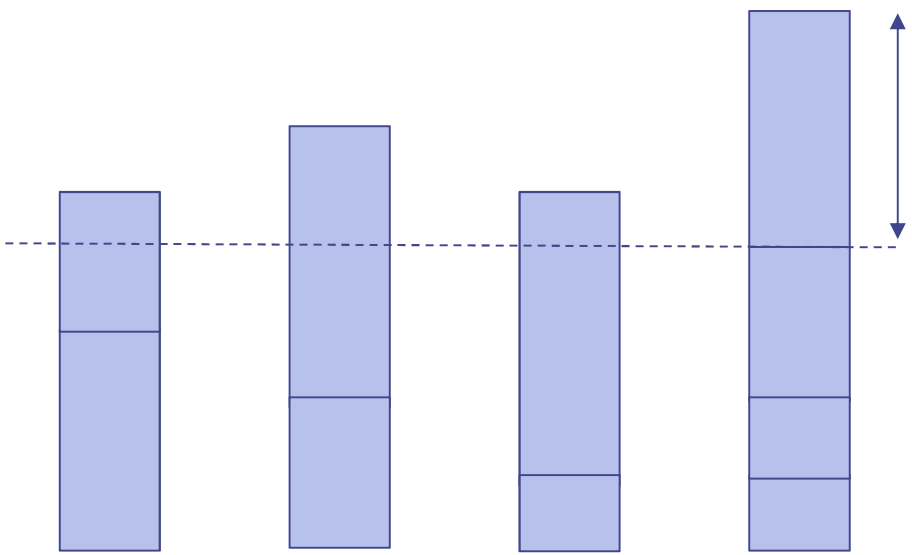
In questo caso, l'algoritmo produrrebbe una soluzione ottima, ma la funzione lower bound non è abbastanza forte da dimostrarlo.

Questa osservazione suggerisce di considerare un’altra funzione lower bound:

$$T^* \geq \max_j \; t_j$$

Analisi del rapporto di approssimazione

Consideriamo la macchina M_i che ha il *massimo carico* T_i, in base all'assegnamento dell'algoritmo. Se l'ultimo lavoro assegnato a tale macchina non era particolarmente più “pesante” rispetto agli altri, allora non siamo troppo al di sopra del primo lower bound, altrimenti non troppo al di sopra del secondo.



Quando si assegna l'attività j a M_i, la macchina ha il carico minore rispetto alle altre.

Questa è la proprietà chiave; prima dell'assegnamento il carico della macchina era $T_i - t_j$ e poiché in quel momento questo era il carico minore, il carico di ciascuna altra macchina era almeno pari a tale valore.

Analisi del rapporto di approssimazione

Sommando i carichi di tutte le macchine:

$$\sum Tk \geq m(Ti - tj)$$

cioè $$T_i - t_j \leq \frac{1}{m} \sum T_k$$

Poichè ciascuna attività è assegnata a una sola macchina, la sommatoria equivale al carico totale dovuto alle attività

$$\sum T_k = \sum t_j$$

Pertanto, in base alla prima funzione lower bound:

$$T_i - t_j \leq \frac{1}{m} \sum t_j \leq T^* \qquad T_i - t_j \leq T^*$$

Algoritmi di approssimazione greedy: load balancing

Consideriamo ora la parte restante del carico su M_i, ossia il lavoro finale t_j.

Analizziamo questo con la seconda funzione lower bound: $t_j <= T^*$

Mettendo insieme le due disuguaglianze:

$$T_i = (T_i - t_j) + t_j \leq 2T^*$$

che prova che il fattore di approssimazione è 2 essendo il makespan pari a T_i.

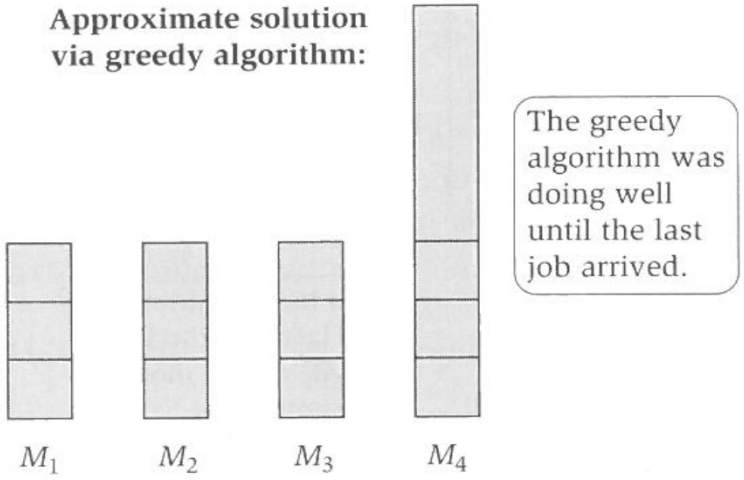
L'algoritmo Greedy-Balance produce un'assegnazione di lavori alle machine con makespan $T \leq 2T^*$

Algoritmi di approssimazione greedy: load balancing

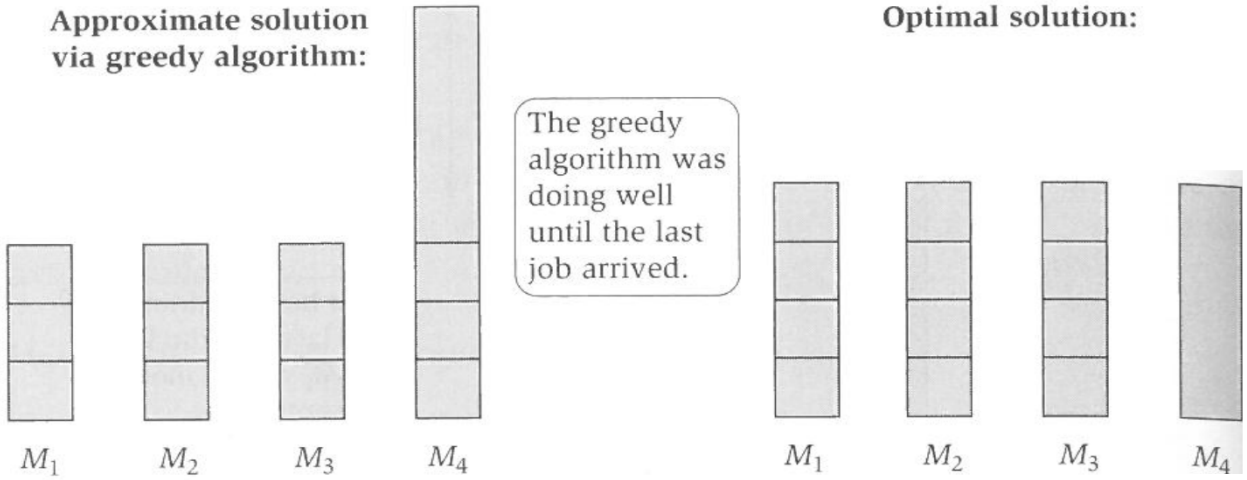
Supponiamo di avere m macchine e che n = m(m-1) +1 sia il numero delle attività. Le prime m(m-1) = n-1 attività richiedono un tempo di esecuzione pari a 1 mentre l'ultima attività richiede un tempo pari a m.

L'algoritmo bilancia le prime n-1 attività, dopodiché deve aggiungere l'ultimo lavoro alla lista di una delle macchine.

Il makespan risultante è T = 2m -1.



Algoritmi di approssimazione greedy: load balancing



Il makespan della soluzione ottima è m. Quindi si ottiene il rapporto di approssimazione

$$(2m-1)/m = 2 - 1/m$$

Il valore si avvicina a 2 al crescere di m.

Nell'esempio il risultato è pessimo, il carico non è bilanciato!

Algoritmi di approssimazione greedy: load balancing

Prendendo spunto dall'esempio precedente, si può pensare di considerare per prime le attività che richiedono più tempo, in modo che l'aggiunta di piccoli lavori futuri possa provocare solamente danni relativamente piccoli.

Infatti, se consideriamo solamente le prime m+1 attività ordinate, ciascuna di esse impiega almeno un tempo pari a t_{m+1}.

- Se ci sono *almeno* m+1 attività e solamente m macchine, ci sarà *almeno una macchina* a cui sono assegnati due o più lavori, e il suo carico sarà *almeno* $2t_{m+1}$.

Si vuole dimostrare che l'algoritmo produce un assegnamento delle attività alle diverse macchine il cui makespan soddisfa la seguente disuguaglianza:

$$T \leq \frac{3}{2} T^*$$

Algoritmi di approssimazione greedy: load balancing

```
Sorted-Balance:
Start with no jobs assigned
Set T_i = 0 and A(i) = ∅ for all machines M_i
Sort jobs in decreasing order of processing times t_j
Assume that t_1 ≥ t_2 ≥ ... ≥ t_n
For j = 1, ..., n
  Let M_i be the machine that achieves the minimum min_k T_k
  Assign job j to machine M_i
  Set A(i) ← A(i) ∪ {j}
  Set T_i ← T_i + t_j
EndFor
```

Se si hanno meno di m attività, la soluzione è chiaramente ottima perché ciascuna attività ha una macchina dedicata.

Se le attività sono più di m, per calcolare il fattore di approssimazione usiamo la seguente funzione lower bound:

$$T^* \geq 2t_{m+1}$$

Analisi del rapporto di approssimazione

Consideriamo, come prima, una macchina M_i, che ha il carico massimo. Se ad essa è assegnata una sola attività, allora l'assegnamento è ottimo.

Assumiamo invece che la macchina abbia almeno due attività distinte da svolgere e che j sia *l'ultimo lavoro assegnatole*; notiamo che j >= m+1 poiché l'algoritmo assegna i primi m lavori a m macchine distinte. Allora:

$$t_j \leq t_{m+1} \leq \frac{1}{2}T^* \qquad \text{essendo} \qquad T^* \geq 2t_{m+1}$$

e mettendo insieme questa disuguaglianza con quella già usata per il primo algoritmo: $T_i - t_j \leq T^*$

si ottiene:

$$T = (T_i - t_j) + t_j \leq T^* + \frac{1}{2}T^* = \frac{3}{2}T^*$$

L'algoritmo Sorted-Balance produce un'assegnazione di job alle machine con makespan T ≤ 3/2T*

Analisi del rapporto di approssimazione

L'algoritmo, detto anche **LPT** (Largest Processing Time), ha complessità **O**(n log n) + **O**(n log m) = **O**(n log n) (ordinamento di n elementi più ricerca del minimo tra m elementi). Con un calcolo più accurato si può dimostrare un limite di:

$$T / T^* <= 4/3 – 1/(3m)$$

Il caso peggiore

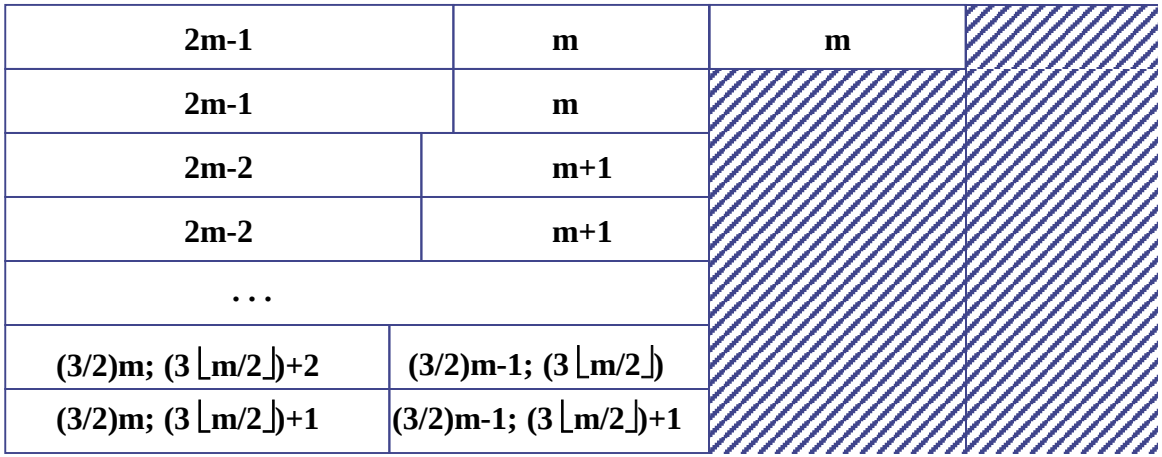
$n = 2m + 1$

$t_1 = t_2 = 2m-1$ . . . $t_{n-4} = t_{n-3} = m + 1$

$t_3 = t_4 = 2m-2$ $t_{n-2} = t_{n-1} = t_n = m$

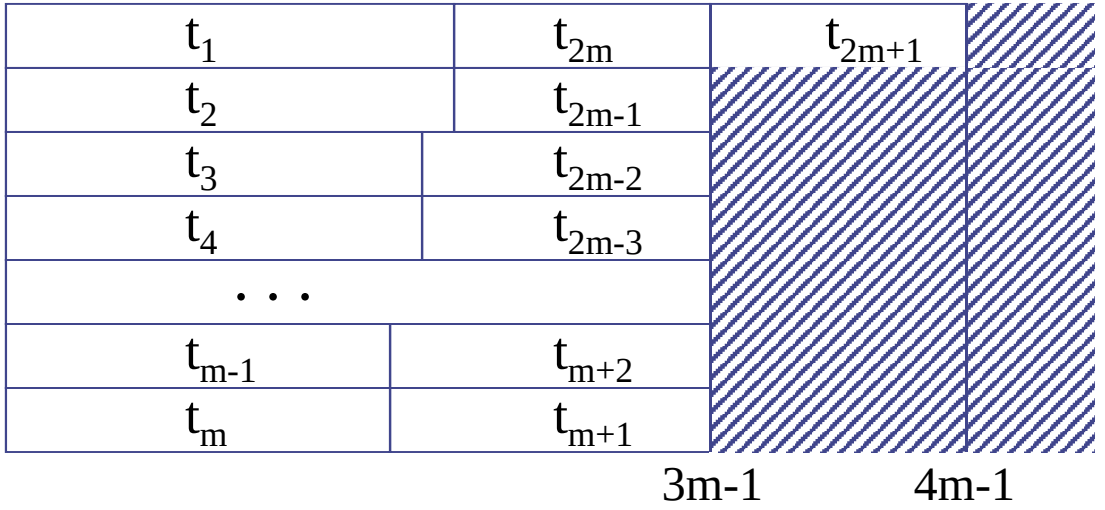
N.B. I *tempi di esecuzione* dei programmi t_{m-1}, t_m, t_{m+1}, t_{m+2}, a seconda che m sia pari o dispari sono rispettivamente:

	m pari	m dispari	
t_{m-1}	(3/2)m	$(3\lfloor m/2 \rfloor)+2$	$t_{m-1} + t_{m+2} = 3m-1$
t_m	(3/2)m	$(3\lfloor m/2 \rfloor)+1$	$t_m + t_{m+1} = 3m-1$
t_{m+1}	(3/2)m-1	$(3\lfloor m/2 \rfloor)+1$	
t_{m+2}	(3/2)m-1	$(3\lfloor m/2 \rfloor)$	

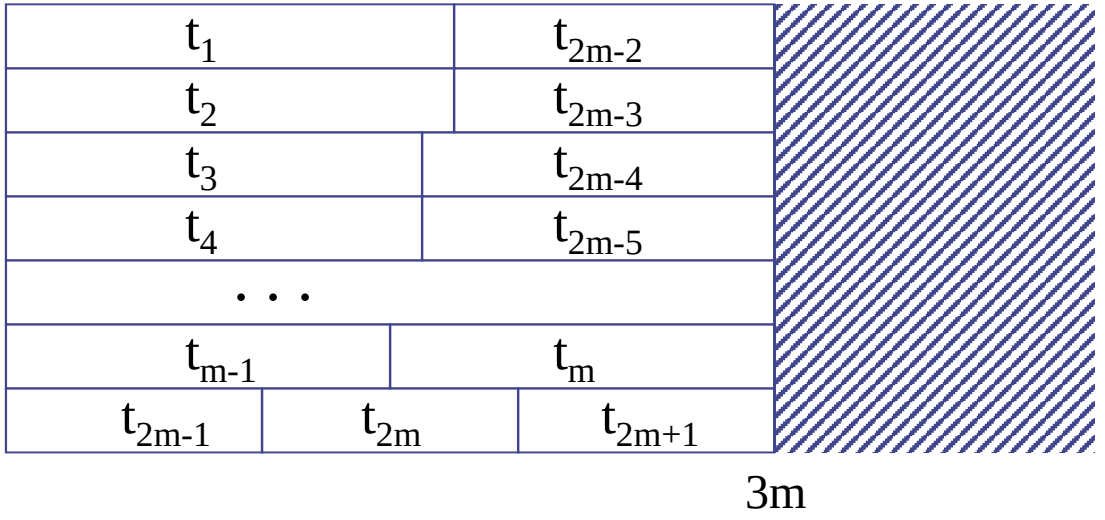


Il caso peggiore

Soluzione approssimata



Soluzione ottima



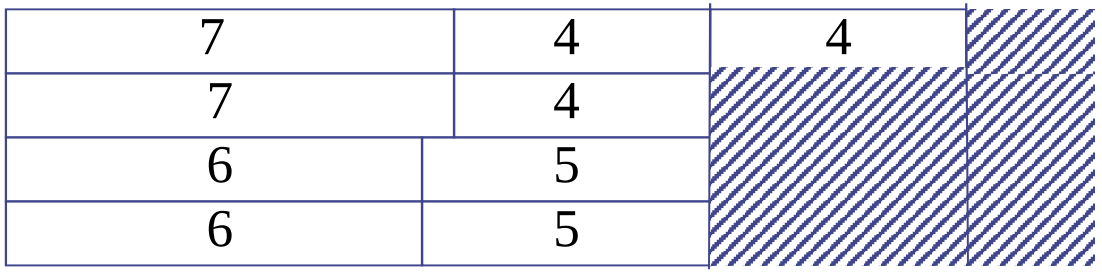
Esempio per m=4

n = 2m + 1= 9

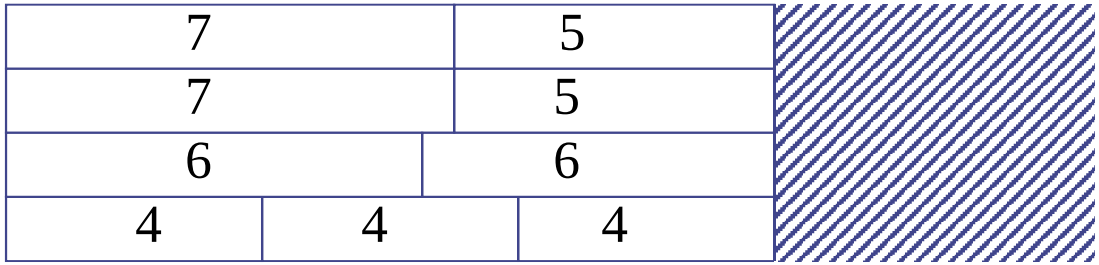
$t_1 = t_2 = 7$
$t_3 = t_4 = 6$

$t_5 = t_6 = 5$
$t_7 = t_8 = t_9 = 4$

Soluzione approssimata



Soluzione ottima



Algoritmi di approssimazione greedy: center selection

Problema: Consideriamo un insieme S di n siti. Vogliamo selezionare k centri per costruire dei grossi centri commerciali. Ciascun individuo, residente in un sito, andrà a fare shopping in uno dei k centri; occorre di conseguenza rendere “bilanciata” la collocazione dei centri, in modo che nessuno di essi sia più sfavorito degli altri.

Considerando i siti come punti del piano, la distanza viene calcolata con la formula di Eulero. In ciascun punto del piano si può collocare un centro.

La funzione per il calcolo della distanza può essere anche diversa, ma deve rispettare i seguenti criteri:

dist(s,s) = 0 per ogni s appartenente a S

dist(s,z) = *dist*(z,s) per ogni s,z appartenente a S

dist(s,z) + *dist*(z,h) ≥ *dist* (s,h) (*disuguaglianza triangolare*)

Algoritmi di approssimazione greedy: center selection

Sia C un insieme di possibili centri. Supponiamo che gli abitanti delle città (i siti s) si rivolgano al centro commerciale più vicino.

$$dist(s, C) = min_{c \in C} dist(s, c)$$

Diciamo che C forma un **r-cover** se dist(s,C) <= r per tutti i siti s in S.

Il minimo valore di r per cui C è un *r-cover* è detto **raggio di copertura** di C e lo indichiamo con *r*(C).

In pratica il raggio di copertura rappresenta il percorso più lungo che ciascuno deve fare per giungere al centro commerciale a lui più vicino.

L'obiettivo è minimizzare *r*(C).

Algoritmi di approssimazione greedy: center selection

Soluzione greedy: selezionare ciascun sito senza considerare dove verranno collocati gli altri, cioe` scegliere il primo centro come se fosse l'unico, ottimizzando la distanza da tutti i siti. Aggiungere nuovi centri tentando di ridurre il più possibile $r(C)$.

Questo approccio però può portare ad assegnamenti poco efficaci. Infatti consideriamo il caso in cui vi siano solo 2 siti (s e z) e k = 2. Chiamiamo d la distanza tra s e z; la scelta migliore per il centro è il punto intermedio tra s e z e, chiamando tale punto c_1, si ha $r(\{c1\}) = d/2$. Se ora aggiungiamo un nuovo centro, qualunque sia la sua posizione sicuramente uno dei due punti sarà ancora più vicino a c_1 e quindi il minimo raggio di copertura sarà ancora $d/2$.

La soluzione ottima si ottiene scegliendo s e z come centri poiché in questo caso si avrebbe $r(C) = 0$.

Algoritmi di approssimazione greedy: center selection

La scelta greedy di mettere il primo centro nel baricentro dei siti perché se ad esempio le città fossero raggruppate in due nuclei metropolitani la soluzione trovata sarebbe molto lontana da quella ottima.

Algoritmi di approssimazione greedy: center selection

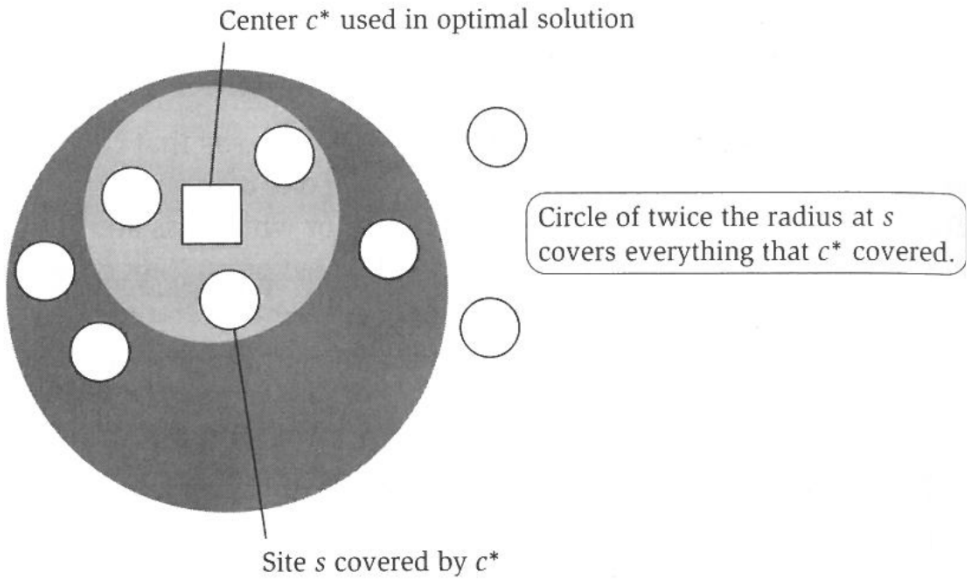
Conoscere il raggio $r(C^*)$ di una soluzione ottima C^*, potrebbe aiutarci ? Si può sfruttare l'esistenza della soluzione C^* per costruire una soluzione approssimata C con raggio di copertura $r(C) \leq 2r(C^*)$

ALGORITMO 1

```
S' will represent the sites that still need to be covered
Initialize S' = S
Let C = ∅
While S' ≠ ∅
   Select any site s ∈ S' and add s to C
   Delete all sites from S' that are at distance at most 2r from s
EndWhile
If |C| ≤ k then
   Return C as the selected set of sites
Else
   Claim (correctly) that there is no set of k centers with
      covering radius at most r
EndIf
```

Analisi del rapporto di approssimazione

Per ogni sito s deve esserci un centro in c* in C* che dista da s al più *r*. Se si sceglie s come centro c, s copre tutti i siti coperti da c* con raggio al massimo 2*r* (disuguaglianza triangolare)



Un insieme di centri C selezionati dall'algoritmo ha raggio di copertura $r(C) \leq 2r(C^*)$

Algoritmi di approssimazione greedy: center selection

- Supponiamo che l'algoritmo fallisca ritornando un insieme di cardinalità maggiore di k.
- Poiché l'algoritmo seleziona come centri i siti, ogni centro c selezionato dall'algoritmo dista da un centro ottimo c* al massimo r : $dist(c,c^*) \leq r$. Diciamo che c* è *prossimo* a c.
-Non può esserci un centro c* prossimo a due siti c e c' selezionati dall'algoritmo. Se no si avrebbe

$$dist(c,c^*) + dist(c^*,c') \leq dist(c,c') < 2r.$$

- Allora C* deve avere ≥ elementi di C e abbiamo

$$k < |C| \leq |C^*|$$ contro l'ipotesi

Se l'algoritmo seleziona più di k centri C, per ogni insieme C* di dimensione al massimo k, $r(C^*) > r$

Algoritmi di approssimazione greedy: center selection

Ma k non è noto!

Per ovviare alla mancata conoscenza di *r*, si può procedere con una sequenza di tentativi effettuando una ricerca «binaria», a costo di aumentare di un fattore logaritmico la complessità.

Nell'esempio particolare della selezione dei centri, vi è un modo facile di superare la non conoscenza del raggio, selezionando ripetutamente i siti s come centri e assicurandoci che siano almeno lontani 2*r* da quelli scelti in precedenza.

Algoritmi di approssimazione greedy: center selection

ALGORITMO 2

Greedy-Center-Selection

Assume $k \leq |S|$ (else define $C = S$)
Select any site s and let $C = \{s\}$
While $|C| < k$
    Select a site $s \in S$ that maximizes $dist(s, C)$
    Add site s to C
EndWhile
Return C as the selected set of sites

In questa soluzione, i centri commerciali sono costruiti nei siti stessi e non in punti qualunque del piano.

Complessità: O(k·n)

Analisi del rapporto di approssimazione

Facciamo vedere che se C* è una soluzione ottima per la collocazione di k con raggio r(C*) centri allora l'algoritmo 2 restiuisce una copertura di k centri con raggio $\leq 2r$

- Notiamo che nell'algoritmo **1** ogni nuovo centro viene selezionato *a caso* tra quelli rimasti, che hanno distanza > 2r da tutti i siti in S'. Finché vi sono ancora siti in S', anche la scelta del sito c' più lontano da tutti i siti già scelti, effettuata dall'algoritmo **2,** è corretta rispetto all'algoritmo **1**.
- Quindi l'algoritmo 2 simula una corretta esecuzione dell'algoritmo 1 e quindi dopo k iterazioni fornisce una copertura C con $r(C) \leq 2r$

L'algoritmo Greedy-Center-Selection seleziona
un insieme C di k centri di raggio
$r(C) \leq 2r(C^*)$, con C* insieme ottimo di k centri.

Algoritmi di approssimazione greedy: set cover

Problema

- Siano dati un insieme finito U e una famiglia F di sottoinsiemi di U tale che ogni elemento di U appartiene ad almeno un sottoinsieme di F : $U = \bigcup_{S \in F} S$
- Una collezione C di sottinsiemi della famiglia F è una copertura di U se:

$$U = \bigcup_{S \in C} S$$

- Il problema "copertura di un insieme" nella forma di problema di ottimizzazione consiste nel trovare una collezione C di cardinalità minima.
- Quindi il costo di una soluzione ammissibile C è |C|.

Algoritmi di approssimazione greedy: set cover

Molti problemi algoritmici possono essere formulati come casi speciali del problema «copertura di insiemi».

Si consideri un insieme $U = \{x_1, ..., x_n\}$ di categorie lavorative (metalmeccanici, autoferrotranvieri...) ed un insieme $F = \{S_1, ..., S_l\}$ di rappresentanti sindacali, ognuno dei quali puo´ rappresentare differenti categorie allo stesso momento.
Ogni elemento S_i di F e` quindi un sottoinsieme di U.

Il problema del Set Covering consiste nel convocare il minor numero possibile di rappresentanti sindacali, in modo tale che ogni categoria lavorativa sia rappresentata da almeno un sindacalista.

Algoritmi di approssimazione greedy: set cover

Soluzione greedy: costruire la copertura un insieme alla volta scegliendo ad ogni passo un insieme che sembra portare più vicino al goal, cioè un insieme S_i che contiene il massimo numero di elementi non ancora coperti. R e` l'insieme degli elementi ancora da coprire.

```
Greedy-Set_Cover (U, F)
        R ← U
        C ← Φ
        while (R ≠ Φ )
                seleziona un S ∈ F che massimizzi |S ∩ R|
                R ← R - S
                C ← C ∪ {S}
        return C
```

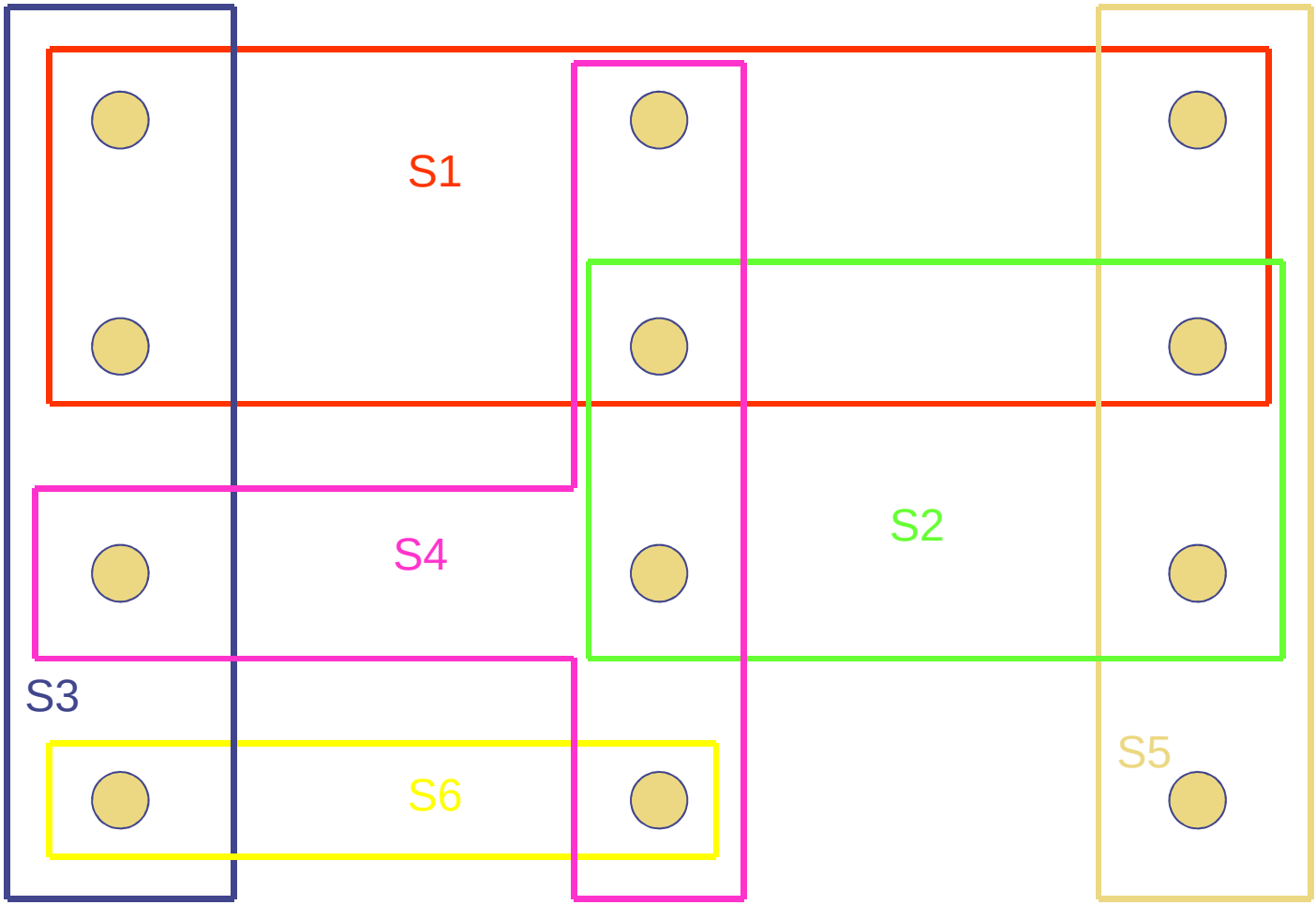
Il corpo del while viene eseguito un numero di volte $\mathbf{O}(\min\{|U|,|F|\})$

La selezione può essere effettuata in tempo $\mathbf{O}(|U|\cdot|F|)$

Complessità: $\mathbf{O}(|U| \cdot |F| \cdot \min\{|U|,|F|\})$

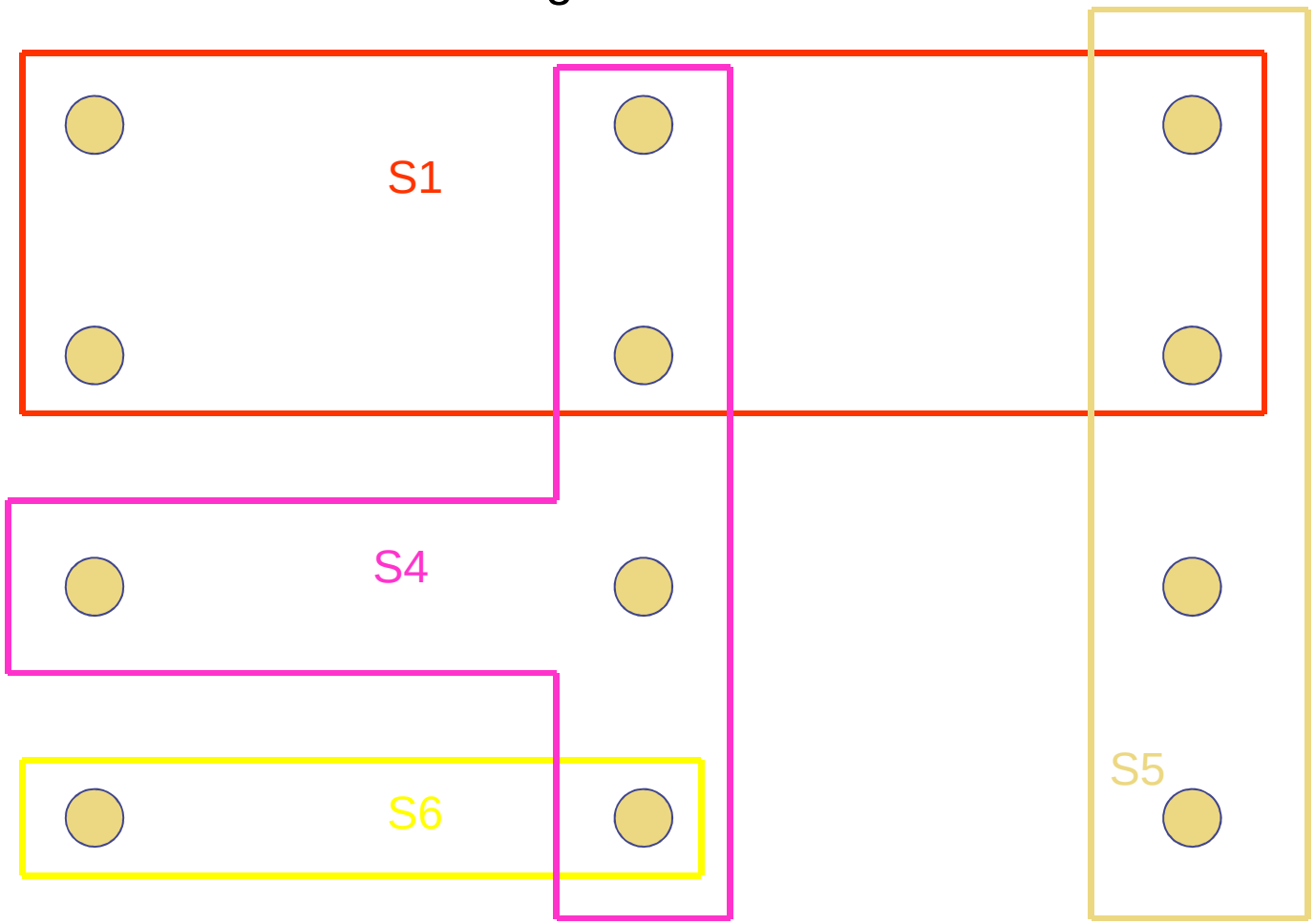
Algoritmi di approssimazione greedy: set cover

Un esempio



Algoritmi di approssimazione greedy: set cover

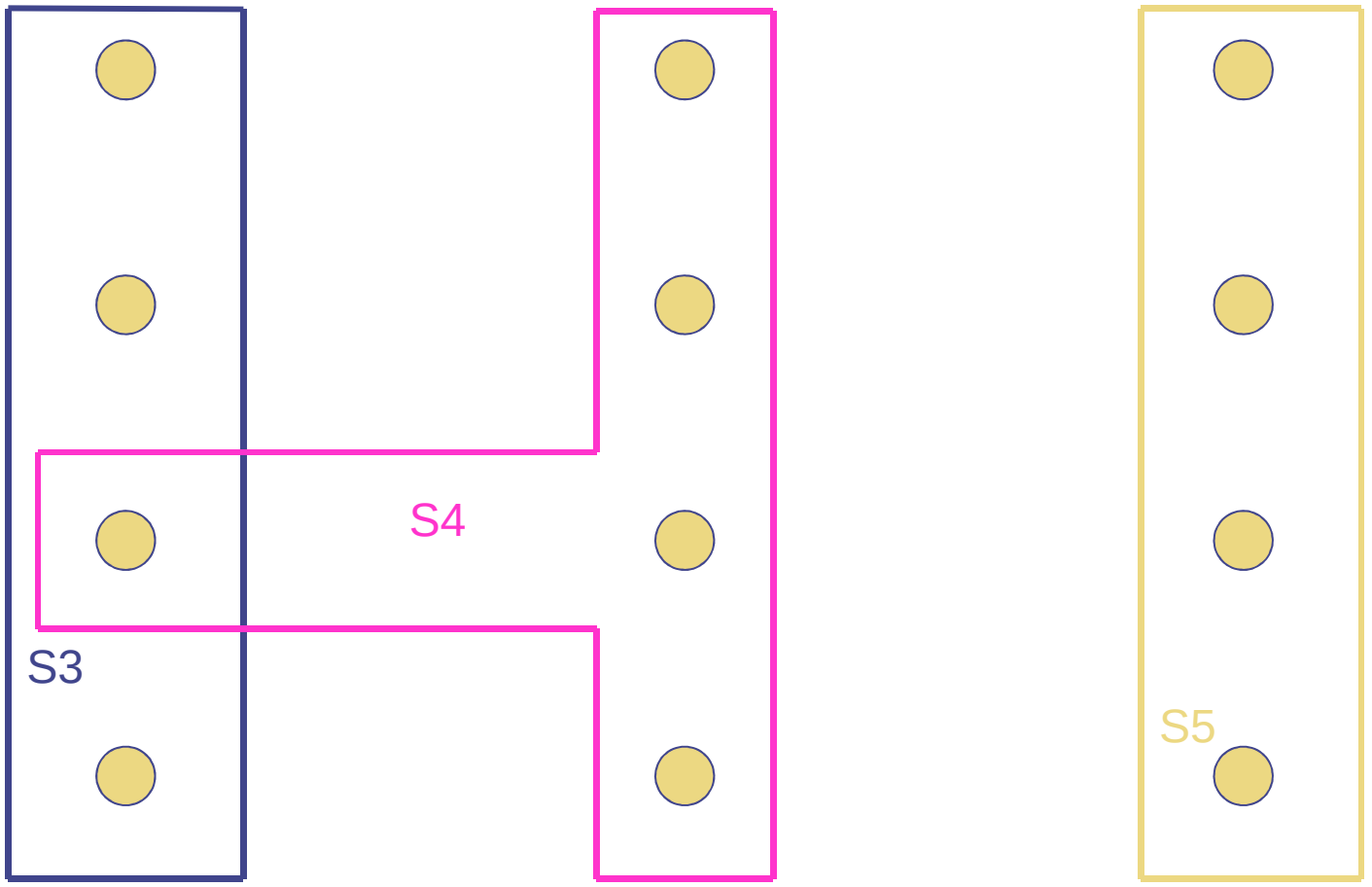
Soluzione ottenuta con l'algoritmo



Soluzione approssimata: 4 sottoinsiemi: S_1, S_4, S_5 e S_6

Algoritmi di approssimazione greedy: set cover

Soluzione ottima



Soluzione ottima: 3 sottoinsiemi: S_4, S_5, S_3

Analisi del rapporto di approssimazione

Per trovare il fattore di approssimazione distribuiamo il costo (unitario) di ogni sottinsieme della soluzione C sugli elementi che esso aggiunge alla copertura.

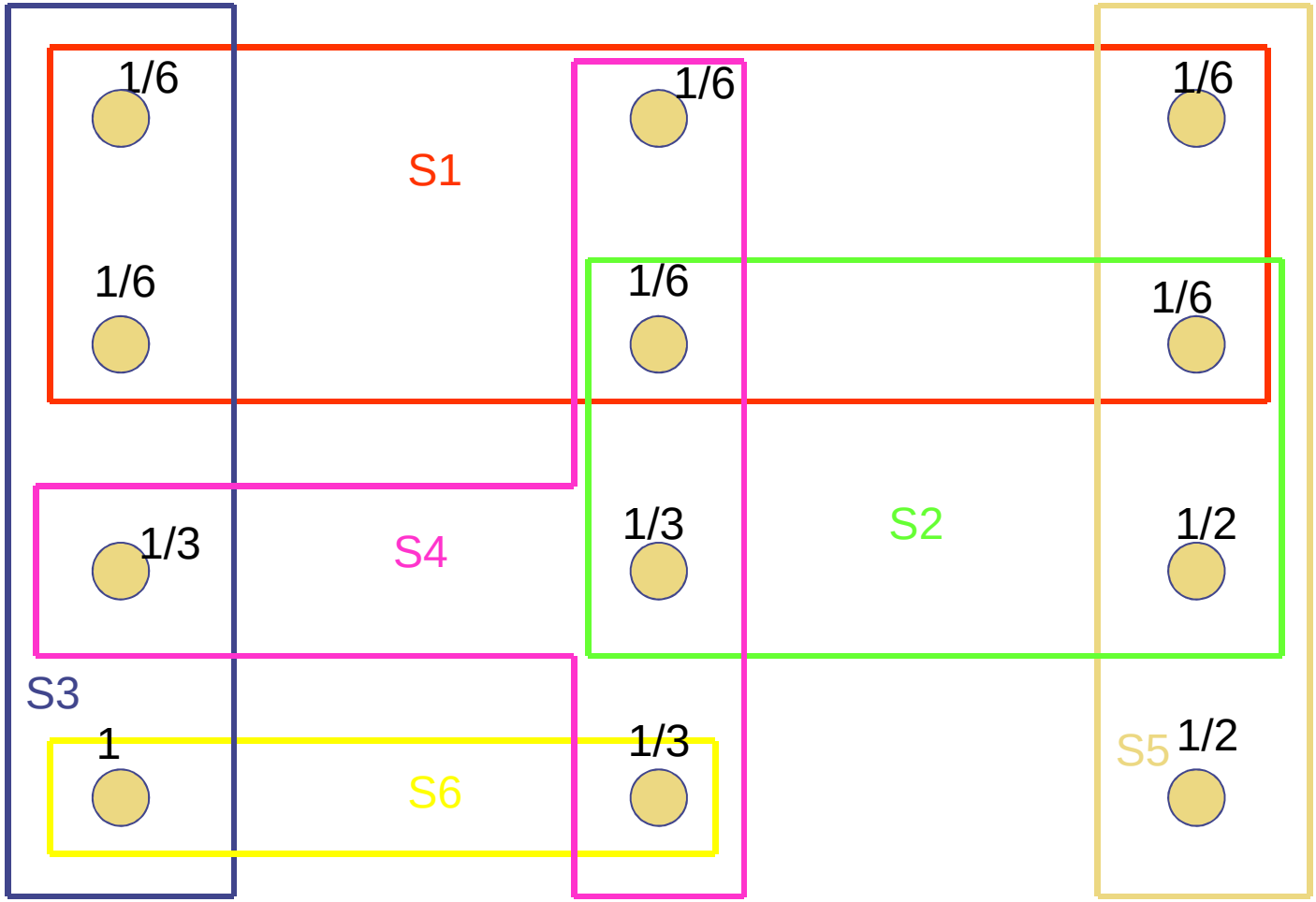
Nell'esempio l'insieme S_1 aggiunge alla copertura 6 nuovi elementi, e quindi il peso di ogni singolo elemento aggiunto alla copertura da S_1 sara` 1/6 , il peso degli elementi aggiunti da $S_4$ sara` 1/3 e cosı` via. Ad ogni elemento dell'insieme *U* viene dunque assegnato un peso p_j , j = 1, ..., n.
Per definizione il valore della soluzione |C| sara` dato da:

$$|C| = \sum_{j \in U} p_j$$

Sia S_i l'i-esimo sottinsieme selezionato dall'algoritmo; il costo attribuito all'elemento j, coperto per la prima volta da S_i, e`:

$$p_j = \frac{1}{|S_i - (S_1 \cup S_2 \ldots \cup S_{i-1})|}$$

Analisi del rapporto di approssimazione



Analisi del rapporto di approssimazione

Si consideri ora la soluzione ottima C*. Essa, essendo una soluzione ammissibile, comprende tutti gli elementi di *U* almeno una volta. Si ha dunque che:

$$|C| = \sum_{j \in U} p_j \leq \sum_{S \in C^*} \sum_{j \in S} p_j$$

poiche´ uno stesso elemento di *U* puo` appartenere a piu` elementi di C*.

Sia S un qualsiasi elemento di *F*. Vogliamo dimostrare che il peso indotto dalla soluzione C dell'algoritmo Greedy-Set-Cover soddisfa la seguente disuguaglianza:

$$\sum_{j \in S} p_j \leq H(|S|)$$

dove H(n) e` la serie armonica arrestata ad n, cioe`

$$H(n) = \sum_{j=1\ldots n} 1/j = 1 + 1/2 + 1/3 + \ldots. + 1/n$$

Analisi del rapporto di approssimazione

Sia $u_i = |S - (S_1 \cup ... \cup S_i)|$ il numero di elementi di S non ancora coperti dopo che gli insiemi S_1, ...S_i sono stati scelti dall'algoritmo. (u_0 è pertanto |S|). La differenza $u_{i-1} - u_i$ è quindi il numero di elementi di S scelti nell'iterazione i-esima, coperti per la prima volta da S_i. Ovviamente $u_{i-1} \geq u_i$. Sia k la prima iterazione in cui $u_k = 0$, ovvero nella quale sono stati scelti gli ultimi elementi di S rimasti scoperti.

$$\sum_{j \in S} p_j = \sum_{i=1...k} \frac{u_{i-1} - u_i}{|S_i - (S_1 \cup S_2 \ldots \cup S_{i-1})|}$$

$$\leq \sum_{i=1...k} \frac{u_{i-1} - u_i}{|S - (S_1 \cup S_2 \ldots \cup S_{i-1})|}$$

Infatti se l'algoritmo greedy sceglie un insieme S_i, esso ricopre un numero maggiore di elementi di S, e quindi il denominatore con S al posto di S_i e` piu` piccolo. Si ha dunque la seguente catena di disuguaglianze:

Analisi del rapporto di approssimazione

$$\sum_{j \in S} p_j \leq \sum_{i=1\ldots k} \frac{u_{i-1} - u_i}{|S - (S_1 \cup S_2 \ldots \cup S_{i-1})|}$$

$$= \sum_{i=1\ldots k} (u_{i-1} - u_i) \frac{1}{u_{i-1}} \qquad (u_{i-1} = |S - (S_1 \cup \ldots \cup S_{i-1})|)$$

$$= \sum_{i=1\ldots k} \sum_{j=u_i+1\ldots u_{i-1}} \frac{1}{u_{i-1}} \qquad (*)$$

$$\leq \sum_{i=1\ldots k} \sum_{j=u_i+1\ldots u_{i-1}} \frac{1}{j} \qquad (\text{infatti } j <= u_{i-1})$$

$$= \sum_{i=1\ldots k} \left(\sum_{j=1\ldots u_{i-1}} \frac{1}{j} - \sum_{j=1\ldots u_i} \frac{1}{j} \right)$$

$$= \sum_{i=1\ldots k} \left(H(u_{i-1}) - H(u_i) \right) = H(u_0) - H(u_k)$$ (gli altri termini si annullano a vicenda)

$$= H(u_0) - H(0) = H(u_0) = H(|S|)$$

* Infatti (ricordiamo che $u_{i-1} \geq u_i$)

$$u_{i-1} - u_i = \sum_{j=u^{i+1},\ldots,u^{i-1}} 1$$

Analisi del rapporto di approssimazione

Sappiamo che $|C| = \sum_{j \in U} p_j \leq \sum_{s \in C^*} \sum_{j \in S} p_j$

e, per quanto appena dimostrato,

$$\sum_{j \in S} p_j \leq H(S)$$

Si ottiene cosi`

$$|C| \leq \sum_{s \in C^*} \sum_{j \in S} p_j \leq \sum_{s \in C^*} H(S) \leq |C^*| \cdot H(\max \{|S|: S \in F\})$$

e quindi, indicando con S* il sottinsieme di C* di cardinalita` massima,

$$|C| / |C^*| \leq H(\max \{|S|: S \in F\}) = H(|S^*|)$$

Il rapporto di approssimazione dell'algoritmo Greedy-Set-Cover e` $\rho(n) = H(\max \{|S|: S \in F\}) = H(|S^*|)$

Analisi del rapporto di approssimazione

Poiché a priori la cardinalità dell'insieme più grande non è limitata (se non dal numero di elementi di U) e la serie armonica è asintotica al logaritmo di n per n grande, nella pratica Greedy-Set-Cover ha un fattore di approssimazione dell'ordine di ln((|U|)):

$$\mathbf{O}(\ln((|U|))$$

La dimensione della soluzione approssimata cresce in modo logaritmico rispetto alla dimensione di una soluzione ottima.

Dato che la funzione logaritmica cresce piuttosto lentamente, questo algoritmo approssimato può dare buoni risultati.

Algoritmi di approssimazione greedy: weighted set cover

L'algoritmo puo` essere facilmente esteso al caso in cui ogni elemento di *F* abbia un peso (Weighted Set Covering): si cerca la copertura di peso complessivo minimo.

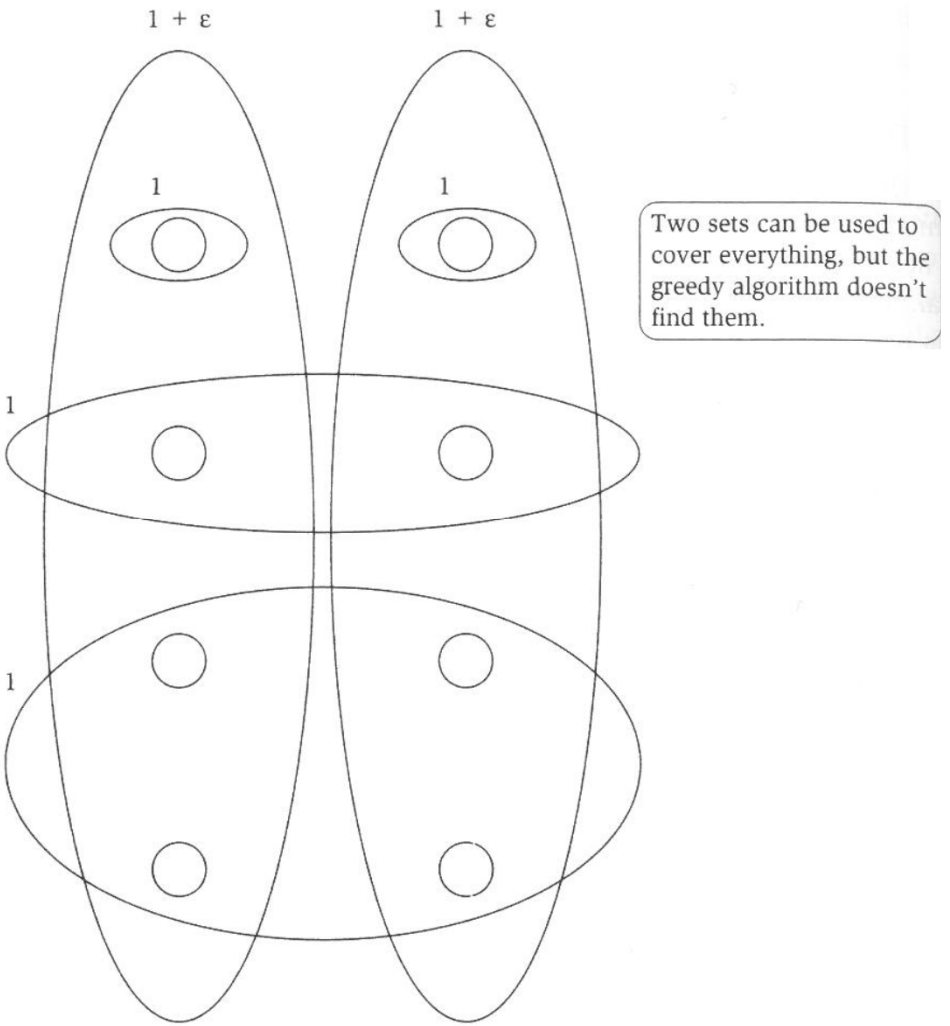
Una soluzione greedy nella scelta del prossimo insieme S_i deve tener conto sia del numero di elementi, sia dei loro pesi.

Le due richieste considerate separatamente non sono sufficienti a fornirci un buon algoritmo, bisogna combinarle in una misura unica considerando anche l'insieme degli elementi non ancora coperti dalla soluzione parziale, cioe` R.
La combinazione dei criteri: $w_i/|S_i \cap R|$ fornisce il costo per elemento coperto.

Algoritmi di approssimazione greedy: weighted set cover

Nell'esempio in figura tutti i sottoinsiemi pesano 1 tranne i due ''verticali'' che pesano 1+ ε

Esaminiamo il comportamento dell'algoritmo: esso costruisce una copertura di peso 4, mentre la copertura ottima ha peso 2 + 2ε



Two sets can be used to cover everything, but the greedy algorithm doesn't find them.

Algoritmi di approssimazione greedy: weighted set cover

Greedy-Weighted-Set-Cover

```
Start with R = U and no sets selected
While R ≠ ∅
  Select set S_i that minimizes w_i/|S_i ∩ R|
  Delete set S_i from R
EndWhile
Return the selected sets
```

Anche per il problema formulato in questo modo si dimostra che il rapporto di approssimazione è H(|S*|), con S* il sottinsieme della famiglia di dimensione massima e quindi dell'ordine del logaritmo di |*U*|.

E' stato dimostrato che nessun algoritmo di approssimazione in tempo polinomiale può raggiungere un rapporto di approssimazione molto migliore di H(|S*|, a meno che non sia **P** = **NP**.