

Laboratorio di Algoritmi

Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano



Lezioni: Martedì 8.30 - 10.30 in aula 3 Mercoledì 10.30 - 13.30 in aula 2
Giovedì 15.30 - 18.30 in aula 2 Venerdì 10.30 - 12.30 in aula 3

Ricevimento: su appuntamento (Dipartimento di Informatica)

Tel.: 02 503 16235

E-mail: roberto.cordone@unimi.it

Pagina web: <http://homes.di.unimi.it/~cordone/courses/2020-algo/2020-algo.html>

Nei problemi di Ottimizzazione Combinatoria

- una **soluzione** x è un **sottoinsieme** di un dato insieme base B finito
- le soluzioni appartengono a una famiglia $X \subseteq 2^B$ di sottoinsiemi che soddisfano opportune condizioni
- la **funzione obiettivo** $f : X \rightarrow \mathbb{N}$ dà un valore a ogni soluzione

Si tratta di **trovare una soluzione di valore minimo o massimo in X**

Algoritmo esaustivo: trova una soluzione ottima ma è esponenziale

- alcuni problemi di *OC* ammettono algoritmi polinomiali esatti
- tutti i problemi di *OC* ammettono algoritmi polinomiali **euristici**, cioè che **non garantiscono di trovare l'ottimo su ogni istanza**

Gli algoritmi greedy

Un **algoritmo greedy** A aggiorna ad ogni passo t un sottoinsieme $x^{(t)}$:

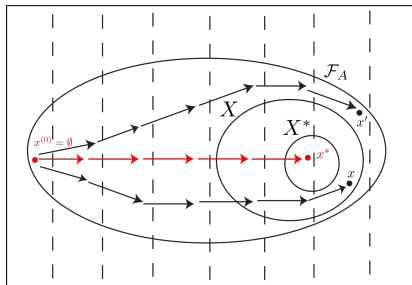
- 1 in $t = 0$ parte da un sottoinsieme vuoto: $x^{(0)} = \emptyset$
(perché ovviamente è parte di una soluzione ottima)
- 2 termina se nessun sottoinsieme più grande può essere ottimo:

$$x \cup \{i\} \notin \mathcal{F}_A \text{ per ogni } i \in B \setminus x$$

\mathcal{F}_A raccoglie i potenziali sottoinsiemi di soluzioni ottime
(potenziali, non sempre sicuri)

- 3 fra gli elementi $i \in B \setminus x$ tali che $x \cup \{i\} \in \mathcal{F}_A$
sceglie l'elemento $i^{(t)}$ che ottimizza un criterio $\phi_A(i, x)$
(tiene x "ammissibile" e cerca di tenerlo "ottimo")
- 4 aggiunge $i^{(t)}$ al sottoinsieme corrente $x^{(t)}$: $x^{(t+1)} := x^{(t)} \cup \{i^{(t)}\}$
(non si torna più indietro nella scelta!)
- 5 torna al punto 2

Per alcuni problemi, trova soluzioni ottime; per altri no

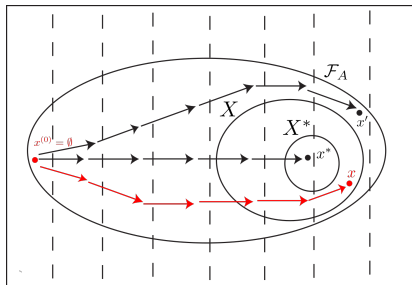


L'algoritmo visita una catena di sottoinsiemi $\emptyset = x^{(0)} \subset \dots \subset x^{(k)}$

Può terminare

- in una soluzione ottima $x^* \in X^*$
- in una soluzione ammissibile non ottima $x \in X$
- in un sottoinsieme non ammissibile x'

Esempi: *albero ricoprente minimo, zaino unitario*

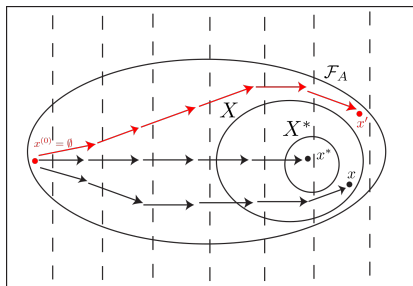


L'algoritmo visita una catena di sottoinsiemi $\emptyset = x^{(0)} \subset \dots \subset x^{(k)}$

Può terminare

- in una soluzione ottima $x^* \in X^*$
- in una soluzione ammissibile non ottima $x \in X$
- in un sottoinsieme non ammissibile x'

Esempio: zaino generico



L'algoritmo visita una catena di sottoinsiemi $\emptyset = x^{(0)} \subset \dots \subset x^{(k)}$

Può terminare

- in una soluzione ottima $x^* \in X^*$
- in una soluzione ammissibile non ottima $x \in X$
- in un sottoinsieme non ammissibile x'

L'algorithmo greedy base

L'algorithmo *greedy* più semplice è quello in cui

- la funzione obiettivo è additiva: $f(x) = \sum_{i \in x} \phi_i$
e non negativa: $\phi_i \geq 0$ per ogni $i \in B$
- si sceglie l'elemento ammissibile che produce il sottoinsieme migliore

$$i^* = \arg \max_{i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A} f(x \cup \{i\}) = \arg \max_{i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A} [f(x) + \phi_i]$$

cioè quello di valore massimo: $i^* = \arg \max_{i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A} \phi_i$

Algorithm Greedy(I)

$x := \emptyset;$

While $\exists i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A$ *do*

$i^* := \arg \max_{i \in B \setminus x : x \cup \{i\} \in \mathcal{F}_A} \phi_i;$

$x := x \cup \{i^*\};$

Return $x;$ { La soluzione migliore visitata è l'ultima }

Il problema dello zaino unitario

Si vuole scegliere da un insieme di oggetti **di pari volume** un sottoinsieme di valore massimo che possa stare in uno zaino di capacità limitata

In questo caso speciale del *KP* il **vincolo di volume diventa di cardinalità**

\mathcal{F}_A coincide con la regione ammissibile $X = \{x \subseteq B : |x| \leq \lfloor V/v \rfloor\}$

Algorithm GreedyUKP(I)

$x := \emptyset;$

While $|x| < \lfloor V/v \rfloor$ *do* { si pone $\mathcal{F}_A = X$ }

$i := \arg \max_{i \in B \setminus x} \phi_i;$

$x := x \cup \{i\};$

Return $x;$

Lo pseudocodice è semplificato dal fatto che x è **estendibile**

- per ogni x di cardinalità $|x| < \lfloor V/v \rfloor$
- aggiungendo qualsiasi elemento $i \in B \setminus x$

Esempio: il problema dello zaino unitario

B		a	b	c	d	e	f
ϕ		7	2	4	5	4	1

$v_i = 1$ per ogni $i \in B$

$$V = 4$$

L'algorithmo esegue i seguenti passi:

- 1 $x := \emptyset$;
- 2 poiché $|x| = 0 < 4$, valuta $i := a$ e aggiorna $x := \{a\}$;
- 3 poiché $|x| = 1 < 4$, valuta $i := d$ e aggiorna $x := \{a, d\}$;
- 4 poiché $|x| = 2 < 4$, valuta $i := c$ e aggiorna $x := \{a, c, d\}$;
- 5 poiché $|x| = 3 < 4$, valuta $i := e$ e aggiorna $x := \{a, c, d, e\}$;
- 6 poiché $|x| = 4 \not< 4$, termina

Questo algorithmo trova sempre la soluzione ottima

Ma perché?

Il problema dello zaino

Si vuole scegliere da un insieme di oggetti **di vario volume** un sottoinsieme di valore massimo che possa stare in uno zaino di capacità limitata

La differenza fondamentale è che si complica la definizione di \mathcal{F}_A dato che **non tutti gli elementi di $B \setminus x$ estendono x in modo ammissibile**

$$x \cup \{i\} \in \mathcal{F}_A = X \Leftrightarrow \sum_{j \in x} v_j + v_i \leq V$$

Algorithm GreedyKP(I)

$x := \emptyset;$

While $\exists i \in B \setminus x : v_i \leq V - \sum_{j \in x} v_j$ *do*

$i := \arg \max_{i \in B \setminus x : v_i \leq V - \sum_{j \in x} v_j} \phi_i;$

$x := x \cup \{i\};$

Return $x;$

Esempio: il problema dello zaino

B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1

$$V = 8$$

L'algorithmo esegue i seguenti passi:

- 1 $x := \emptyset$;
- 2 poiché $v_i \leq V - \sum_{j \in x} v_j \forall i \in B \setminus x$, sceglie $i := a$ e aggiorna $x := \{a\}$;
- 3 poiché $v_i \leq V - \sum_{j \in x} v_j \forall i \in B \setminus x$, sceglie $i := d$ e aggiorna $x := \{a, d\}$;
- 4 poiché $v_i > V - \sum_{j \in x} v_j \forall i \in B \setminus x$, termina

Questo algoritmo non ha trovato la soluzione ottima $x^* = \{a, c, e\}$

Ma perché?

Correttezza dell'algoritmo greedy

Dato un problema di Ottimizzazione Combinatoria con

- insieme base B
- spazio di ricerca (“collezione di indipendenti”) $\mathcal{F}_A \subseteq 2^B$

l'algoritmo greedy lo risolve per ogni funzione obiettivo additiva

$f(x) = \sum_{i \in x} \phi_i$ se e solo se

- ① il sottoinsieme vuoto è un indipendente: $\emptyset \in \mathcal{F}_A$
- ② ogni sottoinsieme proprio di un indipendente è un indipendente: se $x \in \mathcal{F}_A$ e $y \subset x$ allora $y \in \mathcal{F}_A$
- ③ ogni indipendente si può allargare con un opportuno elemento di qualsiasi altro indipendente di cardinalità superiore:
per ogni $x, y \in \mathcal{F}_A$ con $|x| = |y| + 1$, $\exists i \in x \setminus y : y \cup \{i\} \in \mathcal{F}_A$

Queste condizioni

- valgono per il KP unitario
- non valgono per il KP generico

Algoritmi greedy euristici: il KP

Se lo spazio di ricerca \mathcal{F}_A non ha le proprietà adatte, si può adottare

- una definizione sofisticata del criterio di scelta:

$$i = \arg \max_{i \in B \setminus x: x \cup \{i\} \in \mathcal{F}_A} \phi_i$$

diventa

$$i = \arg \max_{i \in B \setminus x: x \cup \{i\} \in \mathcal{F}_A} \varphi_A(i, x)$$

dove $\varphi_A(i, x)$ dipende sia dall'obiettivo sia dai vincoli del problema

Questo consente risultati efficaci, pur se non dimostrabilmente ottimi

Siccome l'algoritmo greedy base per il KP fallisce a causa del volume degli oggetti, si cercano **oggetti di valore alto e volume basso**

- anziché il valore ϕ_i , si usa il **valore unitario** $\varphi_A(i, x) = \frac{\phi_i}{v_i}$

L'algoritmo risultante tipicamente funziona molto meglio

Esempio: il KP

B	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1
ϕ/v	1.40	0.67	2.00	1.67	4.00	1.00

$$V = 8$$

L'algoritmo esegue i seguenti passi:

- 1 $x := \emptyset$;
- 2 sceglie $i := e$ e aggiorna $x := \{e\}$;
- 3 sceglie $i := c$ e aggiorna $x := \{c, e\}$;
- 4 sceglie $i := d$ e aggiorna $x := \{c, d, e\}$;
- 5 sceglie $i := f$ e aggiorna $x := \{c, d, e, f\}$; (l'oggetto a non ci sta)
- 6 poiché $v_i > V - \sum_{j \in x} v_j$ per ogni $i \in B \setminus x$, termina

La soluzione trovata vale 14, quella ottima è $x^* = \{a, c, e\}$ e vale 15

Istanze critiche del KP per l'algoritmo greedy

Nota: la parte che segue è fuori del programma del corso

Ci sono ancora casi critici

B	a	b
ϕ	10	90
v	1	10
ϕ/v	10	9

$$V = 10$$

L'algoritmo esegue i seguenti passi:

- 1 $x := \emptyset$;
- 2 sceglie $i := a$ e aggiorna $x := \{b\}$;
- 3 poiché $v_i > V - \sum_{j \in x} v_j$ per ogni $i \in B \setminus x$, termina

La soluzione vale 10, quella ottima vale 90

L'errore diventa grande a piacere quando
il primo oggetto scartato ha volume grande e valore grande

Un algoritmo 2-approssimato per il KP

Con una piccola modifica, l'algoritmo diventa 2 – approssimato

Il suo risultato è almeno metà dell'ottimo

- 1 Si parte con un sottoinsieme vuoto: $x^{(0)} = \emptyset$
- 2 Si trova l'oggetto i di valore unitario massimo in $B \setminus x$
- 3 Se non eccede il volume, si mette in soluzione e si torna al punto 2

$$x^{(t-1)} = \{i_1, i_2, \dots, i_{t-1}\} \rightarrow x^{(t)} = \{i_1, i_2, \dots, i_t\}$$

- 4 Altrimenti, si costruisce una soluzione col solo oggetto

$$x' = \{i_t\}$$

- 5 Si restituisce la soluzione migliore fra x e x' : $f_A = \max[f(x), f(x')]$

È facile dimostrare che

- la somma delle due soluzioni è una stima per eccesso dell'ottimo

$$f(x) + f(x') = \sum_{\tau=1}^t \phi_{i_\tau} \geq f^*$$

- la migliore delle due soluzioni è almeno metà della somma

$$f_A = \max[f(x), f(x')] \geq \frac{f(x) + f(x')}{2} \geq \frac{1}{2}f^*$$

Il problema dell'albero ricoprente minimo

Dati

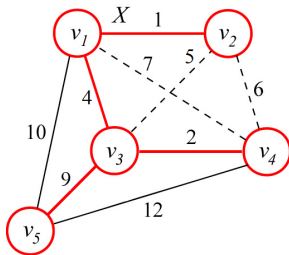
- un grafo non orientato connesso $G = (V, E)$ con $n = |V|$ vertici e $m = |E|$ lati
- una funzione di costo $c : E \rightarrow \mathbb{N}$ definita sui lati

si trovi un sottografo $T^* = (U^*, X^*)$

- 1 ricoprente: U^* contiene tutti i vertici ($U^* = V$)
- 2 connesso: X^* include un cammino fra ogni coppia di vertici u e v
- 3 aciclico: X^* non contiene cicli
- 4 di costo totale minimo:

$c_{X^*} \leq c_X$ per ogni $T = (U, X)$ che soddisfa le proprietà 1, 2 e 3

$$\text{dove } c_X = \sum_{e \in X} c_e$$



Kruskal(V, E, c)

$X := \emptyset$;

$E' := E$;

While $E' \neq \emptyset$

$e^* := \arg \min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\}$;

If **Aciclico**($X \cup \{e^*\}$)

then $X := X \cup \{e^*\}$;

Return (V, X);

$e^* = (v_1, v_2)$

$e^* = (v_3, v_4)$

$e^* = (v_1, v_3)$

$e^* = (v_2, v_3)$ chiude un ciclo

$e^* = (v_2, v_4)$ chiude un ciclo

$e^* = (v_1, v_4)$ chiude un ciclo

$e^* = (v_3, v_5)$

Ogni albero ricoprente ha $n - 1$ lati: STOP

Complessità temporale

Gestire E' come *min-heap*

- costruirlo: $\Theta(m)$
- estrarre il minimo: $\Theta(1)$
- aggiornarlo: $\Theta(\log m)$

Kruskal(V, E, c)

$X := \emptyset;$

$E' := E;$

While $|X| < |V| - 1$

$e^* := \arg \min_{e \in E'} c_e$

$E' := E' \setminus \{e^*\};$

If **Aciclico**($X \cup \{e^*\}$)

then $X := X \cup \{e^*\};$

Return (V, X);

Gestire X come *foresta con bilanciamento*

- costruirla: $\Theta(n)$
- trovare le componenti di u^* e v^* : $\approx \Theta(1)$
- unire le componenti: $\Theta(1)$

Kruskal(V, E, c)

$X := \emptyset;$

$E' := E;$

$E' := \text{CreaHeap}(E');$

$F := \text{CostruisceForesta}(X);$

While $|X| < |V| - 1$

$(u^*, v^*) := \text{ArgMin}(E');$

$E' := \text{CancellaMinimo}(E');$

If **Find**(u^*, F) \neq **Find**(v^*, F)

then $X := X \cup \{(u^*, v^*)\};$

$\text{Union}(F_{u^*}, F_{v^*}, F);$

Return (V, X);

$\Theta(1)$

$\Theta(1)$

$\Theta(m)$

$\Theta(n)$

$i_{\max} \leq m$

$\Theta(1)$

$\Theta(\log n)$

$\approx \Theta(1)$

$\Theta(1)$

$\Theta(1)$