Programmazione dinamica

La *programmazione dinamica* è un altro approccio che consente di risolvere problemi in modo esatto.

Considereremo solo problemi di massimo ma piccole modifiche consentono di applicare tale approccio anche a problemi di minimo.

La programmazione dinamica è applicabile a problemi con determinate proprietà che andremo ora a elencare.

Proprietà

- Il problema può essere suddiviso in n *blocchi*.
- In ogni blocco k, $k = 1, \ldots, n$ ci si trova in uno degli *stati* s_k appartenenti all'insieme di stati $Stati_k$. L'insieme $Stati_1$ del blocco 1 è costituito da un singolo stato s_1.
- In ogni blocco k si deve prendere una decisione d_k appartenente ad un insieme di possibili decisioni D_k. L'insieme di possibili decisioni può dipendere dallo stato s_k, ovvero $D_k = D_k(s_k)$.

Continua

- Se al blocco k si prende la decisione d_k e ci si trova nello stato s_k, il blocco k fornisce un *contributo* alla funzione obiettivo f del problema pari a $u(d_k, s_k)$. La funzione obiettivo f sarà pari alla somma dei contributi degli n blocchi (esistono anche varianti in cui i contributi non si sommano ma si moltiplicano tra loro ma qui ne omettiamo la trattazione).
- Se al blocco k ci si trova nello stato s_k e si prende la decisione d_k, al blocco $k+1$ ci troveremo nello stato $s_{k+1} = t(d_k, s_k)$. La funzione t viene detta *funzione di transizione*

Il principio di ottimalità

Ma la proprietà essenziale è quella che viene chiamata *principio di ottimalità*:

se al blocco k mi trovo nello stato s_k, la sequenza di decisioni ottime da prendere nei blocchi $k, k+1, \ldots, n$ è **totalmente indipendente** *da come sono giunto allo stato s_k, ovvero dalle decisioni ai blocchi $1, \ldots, k-1$ che mi hanno fatto arrivare allo stato s_k.*

Nel problema dello zaino

- blocchi $\equiv$ oggetti
- al blocco k lo stato s_k rappresenta la capacità residua dello zaino una volta prese le decisioni relative agli oggetti $1, 2, \ldots, k-1$. Quindi gli insiemi di stati possibili in ogni blocco sono:

$$Stati_k = \{0, 1, \ldots, b\} \text{ per } k = 2, \ldots, n, \quad Stati_1 = \{b\}$$

- $D_k(s_k) = \begin{cases} \{\text{NO}\} & \text{se } s_k < p_k \\ \{\text{NO, SI}\} & \text{se } s_k \geq p_k \end{cases}$

Continua

- contributo del blocco k:

$$u(d_k, s_k) = \begin{cases} 0 & \text{se } d_k = \text{NO} \\ v_k & \text{se } d_k = \text{SI} \end{cases}$$

(**NB**: in questo caso il contributo non dipende dallo stato s_k)

- funzione di transizione:

$$t(d_k, s_k) = \begin{cases} s_k & \text{se } d_k = \text{NO} \\ s_k - p_k & \text{se } d_k = \text{SI} \end{cases}$$

Continua

Il principio di ottimalità è soddisfatto:

se ho capacità residua dello zaino s_k, le decisioni ottime per i blocchi $k, k+1, \ldots, n$ si ottengono risolvendo un problema dello zaino con i soli oggetti da k fino a n e con capacità dello zaino s_k e *non dipendono da come sono arrivato ad avere capacità residua s_k nello zaino.*

Ci accorgiamo però della seguente relazione tra una soluzione del nostro problema, e la soluzione dello stesso problema per uno zaino più piccolo (cioè meno resistente): se ho una soluzione ottima (cioè il cui valore è massimo tra tutte le soluzioni possibili) per uno zaino che regge P, e tolgo dalla soluzione un oggetto qualsiasi di tipo t, ottengo una soluzione ottima per uno zaino che regge $P - p(t)$. Infatti, se per assurdo esistesse una soluzione migliore con peso inferiore a $P - p(t)$, potrei aggiungerle un oggetto di tipo t e ottenere così una soluzione migliore per il problema originale (il che è impossibile, avendo assunto che la soluzione fosse ottima).

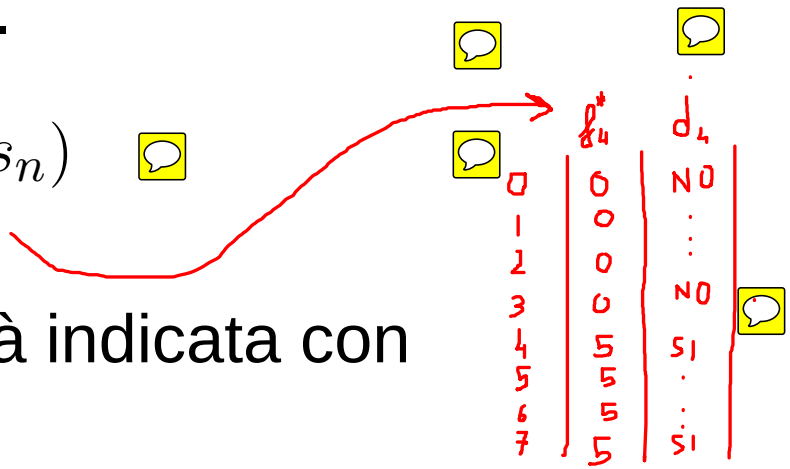
Questo significa che se conosciamo la soluzione ottima per uno zaino di peso Q, possiamo ottenere nuove soluzioni per zaini di grandezza superiore aggiungendo un oggetto di tipo t, per ogni tipo t in T. In particolare, se ho una soluzione di valore V per uno zaino di peso Q, allora so che esiste una soluzione di valore $V + v(t)$ per uno zaino di peso $P + p(t)$, per ogni t in T. *Tutte le soluzioni ottime si ottengono in questo modo.*

Funzione f_k^*

La funzione $f_k^*(s_k)$ restituisce il valore ottimo delle somme dei contributi dei blocchi $k, k+1, \ldots, n$ quando si parte dallo stato s_k al blocco k.

Tipicamente è semplice calcolare $f_n^*(s_n)$ per ogni $s_n \in Stati_n$, cioè il valore ottimo del solo contributo del blocco n quando ci si trova nello stato s_n.

$$f_n^*(s_n) = \max_{d_n \in D_n(s_n)} u(d_n, s_n)$$



La corrispondente decisione ottima verrà indicata con $d_n^*(s_n)$.

A questo punto posso procedere a ritroso per calcolare $f_{n-1}^*(s_{n-1})$ per ogni $s_{n-1} \in Stati_{n-1}$.

Continua

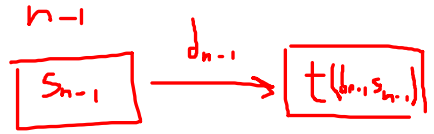
Dato uno stato s_{n-1} in $Stati_{n-1}$, considero una generica decisione $d_{n-1} \in D_{n-1}(s_{n-1})$.

Il contributo di tale decisione al blocco $n-1$ è pari a $u(d_{n-1}, s_{n-1})$.

Inoltre, mi sposto nello stato $s_n = t(d_{n-1}, s_{n-1})$ al blocco n.

Da qui sfrutto il principio di ottimalità in base a cui non importa come sono arrivato nello stato $t(d_{n-1}, s_{n-1})$ e posso quindi procedere in modo ottimo da esso con un contributo pari a $f_n^*(t(d_{n-1}, s_{n-1}))$.

Riassumendo ...



... se mi trovo nello stato s_{n-1} e prendo la decisione d_{n-1} il contributo ottimo complessivo dei blocchi $n-1$ e n sarà dato da

$$u(d_{n-1}, s_{n-1}) + f_n^*(t(d_{n-1}, s_{n-1})).$$

Tra tutte le possibili decisioni d_{n-1} la migliore sarà quella che rende massimo il contributo complessivo.

Tale decisione viene indicata con $d_{n-1}^*(s_{n-1})$ e si avrà

$$f_{n-1}^*(s_{n-1}) = u(d_{n-1}^*(s_{n-1}), s_{n-1}) + f_n^*(t(d_{n-1}^*(s_{n-1}), s_{n-1})) =$$

$$= \max_{d_{n-1} \in D_{n-1}(s_{n-1})} [u(d_{n-1}, s_{n-1}) + f_n^*(t(d_{n-1}, s_{n-1}))].$$

Continua

Una volta calcolati i valori $f^*_{n-1}(s_{n-1})$ per tutti gli stati $s_{n-1} \in Stati_{n-1}$, possiamo continuare a procedere a ritroso.

Per il blocco k se mi trovo nello stato s_k e considero una generica decisione $d_k \in D_k(s_k)$, il contributo di tale decisione al blocco k è pari a $u(d_k, s_k)$.

Inoltre, mi sposto nello stato $s_{k+1} = t(d_k, s_k)$ al blocco $k+1$.

Da qui sfrutto il principio di ottimalità in base a cui non importa come sono arrivato nello stato $t(d_k, s_k)$ e posso quindi procedere in modo ottimo da esso con un contributo pari a $f^*_{k+1}(t(d_k, s_k))$.

Quindi ...

... se mi trovo nello stato s_k e prendo la decisione d_k il contributo ottimo complessivo dei blocchi $k, k+1, \ldots, n$ sarà dato da

$$u(d_k, s_k) + f^*_{k+1}(t(d_k, s_k)).$$

da cui, prendendo la decisione che massimizza tale valore, avremo:

$$f^*_k(s_k) = \max_{d_k \in D_k(s_k)} [u(d_k, s_k) + f^*_{k+1}(t(d_k, s_k))],$$

con la corrispondente decisione ottima $d^*_k(s_k)$.

NB: si noti che poiché si procede a ritroso, i valori f^*_{k+1} sono già stati calcolati.

Valore ottimo

Arrivati al blocco 1 si ha che $Stati_1$ è formato da un unico stato s_1 e il valore $f_1^*(s_1)$ coincide con il valore ottimo del problema.

Soluzione ottima

Per ricostruire la soluzione ottima possiamo partire dal blocco 1:

- al blocco 1 la decisione ottima è $d_1^*(s_1)$ e con tale decisione ci spostiamo allo stato $s_2^* = t(d_1^*(s_1), s_1)$ del blocco 2;
- al blocco 2 la decisione ottima sarà $d_2^*(s_2^*)$. Con tale decisione ci spostiamo allo stato $s_3^* = t(d_2^*(s_2^*), s_2^*)$ del blocco 3;
- al blocco 3 la decisione ottima sarà $d_3^*(s_3^*)$;
- si procede in questo modo fino ad arrivare al blocco n.

Algoritmo- valore ottimo

- **Passo 1** Per ogni $s_n \in Stati_n$ si calcoli $f_n^*(s_n)$ e la corrispondente decisione ottima $d_n^*(s_n)$. Si ponga $k = n - 1$.
- **Passo 2** Per ogni $s_k \in Stati_k$ si calcoli

$$f_k^*(s_k) = \max_{d_k \in D_k(s_k)} [u(d_k, s_k) + f_{k+1}^*(t(d_k, s_k))]$$

  e la corrispondente decisione ottima $d_k^*(s_k)$.
- **Passo 3** Se $k = 1$ si ha che $f_1^*(s_1)$ è il valore ottimo del problema. Altrimenti si ponga $k = k - 1$ e si ritorni al Passo 2.

Algoritmo - soluzione ottima

- **Passo 1** Si ponga $s_1^* = s_1$ e $k = 1$.
- **Passo 2** Per il blocco k la decisione ottima è $d_k^*(s_k^*)$. Si ponga

$$s_{k+1}^* = t(d_k^*(s_k^*), s_k^*).$$

- **Passo 3** Se $k = n$, stop: la soluzione ottima è stata ricostruita ed è rappresentata dalle decisioni

$$d_1^*(s_1^*), \ldots, d_n^*(s_n^*).$$

Altrimenti si ponga $k = k + 1$ e si ritorni al Passo 2.

Nota bene

La programmazione dinamica rientra tra gli algoritmi di enumerazione implicita.

Infatti, pensando al problema dello zaino, una volta risolto il sottoproblema del calcolo del valore $f_k^*(s_k)$, non devo valutare esplicitamente ogni singola soluzione ammissibile che abbia capacità residua s_k in corrispondenza dell'oggetto k ma posso limitarmi a considerare le sole soluzioni ammissibili che scelgono gli oggetti $k, \ldots, n$ in modo ottimale, ovvero in modo da risolvere il sottoproblema dello zaino con i soli oggetti $k, \ldots, n$ e capacità residua s_k.

Complessità

Dato uno stato s_k del blocco k, per ogni decisione $d_k \in D_k(s_k)$ si deve:

- calcolare $u(d_k, s_k)$;
- calcolare $t(d_k, s_k)$;
- calcolare $u(d_k, s_k) + f^*_{k+1}(t(d_k, s_k))$.

Poi, si deve trovare il minimo di tali valori al variare di $d_k \in D_k(s_k)$.

Quindi, nel blocco k il numero di operazioni richieste è:

$$\sum_{s_k \in S_k} 4|D_k(s_k)|$$

Di conseguenza, il numero compelssivo di operazioni è:

$$\sum_{k=1}^{n} \sum_{s_k \in S_k} 4|D_k(s_k)|$$

Nel problema dello zaino

- $|S_1| = 1$, mentre per $k = 2, \ldots, n$ si ha $|S_k| = b + 1$;
- per ogni stato s_k si ha $|D_k(s_k)| = 1$ oppure 2.

Quindi, il numero totale di operazioni è $O(nb)$.

Questa è una complessità esponenziale. Perché?

Il valore b è esponenziale rispetto alla sua codifica in codice binario che richiede $\lceil \log_2(b+1) \rceil$ bit.

Problema schedulazione

Come altro esempio di applicazione della programmazione dinamica, consideriamo un problema di schedulazione.

Supponiamo di avere n job (processi) $J_1, \ldots, J_n$ da eseguire su una macchina, ciascuno con un tempo di esecuzione p_i.

Supponiamo di dover decidere quando eseguire ciascuno di essi in modo da minimizzare la misura di prestazione $\sum_{i=1}^{n} \gamma_i(C_i)$, dove C_i è l'istante in cui termina il job J_i e $\gamma_i(C_i)$ è una funzione non decrescente di C_i.

Si può dimostrare che nella soluzione ottima i job sono schedulati uno dietro l'altro senza pause tra di essi e quindi si può restringere l'attenzione a tutte le possibili permutazioni degli n job (in totale $n!$).

Problema schedulazione

Per certe funzioni γ_i esistono algoritmi di risoluzione molto semplici ed efficienti. Per esempio, per $\gamma_i(C_i) = \frac{C_i}{n}$ (tempo di completamento medio), la permutazione ottima è la permutazione $i(1), \ldots, i(n)$ che soddisfa

$$p_{i(1)} \leq p_{i(2)} \leq \cdots \leq p_{i(n)}.$$

Tale permutazione viene chiamata anche schedulazione Shortest Processing Time (SPT).

Ma in altri casi la permutazione ottima è molto più complicata da trovare. Un esempio è il seguente

$$\gamma_i(C_i) = \frac{1}{n} \max\{0, C_i - d_i\} = \frac{1}{n} T_i,$$

dove T_i misura il ritardo dell'istante di completamento del job J_i rispetto a una data prevista d_i. In tal caso la misura di prestazione è il ritardo medio, indicato con $\bar{T}$.

Osservazione (principio di ottimalità)

Supponiamo che $J_{i(1)}, \ldots, J_{i(n)}$ *sia una schedulazione ottima. Allora, per ogni* $k = 1, \ldots, n$, $J_{i(1)}, \ldots, J_{i(k)}$ *è una schedulazione ottima per il problema ridotto in cui si considerano solo i job* $J_{i(1)}, \ldots, J_{i(k)}$.

Dimostrazione Scomponiamo in questo modo la misura di prestazione

$$\sum_{r=1}^{k} \gamma_{i(r)}(C_{i(r)}) + \sum_{r=k+1}^{n} \gamma_{i(r)}(C_{i(r)}).$$

Continua

Per assurdo, supponiamo esista una schedulazione migliore $J_{i'(1)}, \ldots, J_{i'(k)}$ dei job $J_{i(1)}, \ldots, J_{i(k)}$.

Allora avremmo

$$\sum_{r=1}^{k} \gamma_{i'(r)}(C_{i'(r)}) + \sum_{r=k+1}^{n} \gamma_{i(r)}(C_{i(r)}) < \sum_{r=1}^{k} \gamma_{i(r)}(C_{i(r)}) + \sum_{r=k+1}^{n} \gamma_{i(r)}(C_{i(r)}),$$

ovvero $J_{i'(1)}, \ldots, J_{i'(k)}, J_{i(k+1)}, \ldots, J_{i(n)}$ sarebbe migliore di $J_{i(1)}, \ldots, J_{i(n)}$, contraddicendo dunque l'ottimalità di $J_{i(1)}, \ldots, J_{i(n)}$.

Ottimo del problema ridotto

Consideriamo il problema ridotto in cui si ha solo un sottinsieme Q dei job $J_1, \ldots, J_n$.

Indichiamo con $\Gamma(Q)$ il minimo della misura di prestazione per il problema ridotto.

Se $Q = \{J_i\}$ contiene un solo job, allora

$$\Gamma(\{J_i\}) = \gamma_i(p_i).$$

Se $|Q| > 1$, allora

$$\Gamma(Q) = \min_{J_i \in Q} \Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q),$$

dove $C_Q = \sum_{J_r \in Q} p_r$.

Infatti ...

... se supponiamo che come ultimo job tra quelli in Q venga messo J_i, allora, in base all'osservazione fatta, la misura minima è data dal valore ottimo del problema ridotto con i job $Q \setminus \{J_i\}$ (pari a $\Gamma(Q \setminus \{J_i\})$) a cui si somma il contributo del job J_i pari a $\gamma_i(C_Q)$, in quanto, se il job J_i viene messo per ultimo tra quelli in Q, non essendoci momenti di inattività della macchina, verrà completato all'istante C_Q.

Siccome il job finale deve essere ovviamente uno di quelli in Q, ne consegue che il valore $\Gamma(Q)$ deve essere il minimo, al variare di $J_i \in Q$, dei valori $\Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q)$.

Si noti che quando $Q = \{J_1, \ldots, J_n\}$, il valore $\Gamma(Q)$ è il valore ottimo del nostro problema.

Algoritmo

Passo 1 Poni $\Gamma(\{J_i\}) = \gamma_i(p_i)$, $i = 1, \ldots, n$ e poni $k = 2$.

Passo 2 Per ogni $Q \subseteq \{J_1, \ldots, J_n\}$, con $|Q| = k$, calcola

$$\Gamma(Q) = \min_{J_i \in Q} \Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q).$$

e memorizza

$$J_{i^*(Q)} = \arg \min_{J_i \in Q} \Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q).$$

Passo 3 Se $k = n$, il valore $\Gamma(\{J_1, \ldots, J_n\})$ è il valore ottimo del problema e vai al Passo 4. Altrimenti poni $k = k + 1$ e vai al Passo 2.

Algoritmo

Passo 4 Poni $l = n$ e $\bar{Q} = \{J_1, \ldots, J_n\}$.

Passo 5 Poni

$$J_{i(l)} = J_{i^*(\bar{Q})} \quad \bar{Q} = \bar{Q} \setminus \{J_{i^*(\bar{Q})}\}.$$

Se $l = 1$, STOP, altrimenti poni $l = l - 1$ e ripeti il Passo 5.

Algoritmo programmazione dinamica

L'algoritmo appena descritto è un algoritmo di programmazione dinamica dove:

- il numero di blocchi è pari a n;
- nel blocco k l'insieme degli stati è costituito da tutti i sottinsiemi di job di cardinalità k;
- in corrispondenza di un determinato stato Q, la decisione d_k da prendere è quale membro di Q mettere in ultima posizione;
- se, trovandomi nello stato Q, decido di mettere in ultima posizione $J_i \in Q$, il contributo u di tale decisione è $\gamma_i(C_Q)$;
- se, trovandomi nello stato Q, decido di mettere in ultima posizione $J_i \in Q$, effettuo una transizione verso lo stato $Q \setminus \{J_i\}$ del blocco $k-1$.

Nota bene

Rispetto all'algoritmo visto per il problema dello zaino, qui si procede prima in avanti (dal blocco 1 al blocco n) per individuare il valore ottimo, e poi all'indietro per ricostruire la soluzione ottima.

Complessità algoritmo

Consideriamo solo la complessità della prima parte dell'algoritmo (Passi 1-3), visto che la seconda parte (Passi 4-5) richiede uno sforzo computazionale molto minore.

All'iterazione k dobbiamo considerare tutti i sottinsiemi di k degli n job. Il numero di tali sottinisiemi è

$$\left(\begin{array}{c} n \\ k \end{array} \right).$$

Per ogni sottinsieme si deve: (i) calcolare $\gamma_i(C_Q)$ per ogni $J_i \in Q$; (ii) calcolare $\Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q)$ per ogni $J_i \in Q$; (iii) trovare il minimo dei k valori $\Gamma(Q \setminus \{J_i\}) + \gamma_i(C_Q)$.

Continua

Quindi in tutto abbiamo a ogni iterazione k un numero di operazioni

$$3k \left(\begin{array}{c} n \\ k \end{array} \right) .$$

Sommando su tutte le iterazioni, abbiamo un numero totale di iterazioni pari a

$$\sum_{k=1}^{n} 3k \left(\begin{array}{c} n \\ k \end{array} \right) = O\left(n 2^n\right) .$$

Esempio

Si trovi la schedulazione ottima rispetto a $\bar{T}$ per il seguente problema con $m = 1$

Job	J_1	J_2	J_3	J_4
p_i	8	6	10	7
d_i	14	9	16	16