Risoluzione del Multiple-Choice Knapsack Problem con la Programmazione Dinamica

Elaborato di progetto per il corso di
Metodi ed Algoritmi di Ottimizzazione per il Problem Solving

https://bitbucket.org/chiara-volonnino/maoxps-19-mckp

Chiara Volonnino - Mat. 843650[1]

Email ids: [1]chiara.volonnino@studio.unibo.it

A.A. 2018/2019

Indice

Introduzione

L'obiettivo del progetto é ideare e realizzare in forma prototipale un modello di risoluzione del problema del Multiple-Choice Knapsack tramite lo sfruttamento della **programmazione dinamica**. In particolare si implementerà l'algoritmo risolutivo con il linguaggio di programmazione ad alto livello `C++`.

In questo documento infatti si definisce cos'è *Multiple-Choice Knapsack Problem*, sia in modo informale che in maniera formale.

Si parlerà della programmazione dinamica e di come, grazie a questa é possibile risolvere il problema in questione, in maniera semplice, veloce e raffinata.

Si spiegherà nel dettaglio tutta la procedura implementativa.

In fine si andranno ad confrontare e discutere i risultati ottenuti.

Si andranno poi a discutere alcune varianti del MCKP.

Capitolo 1

Multiple-Choice Knapsack Problem

Il Multiple-Choice Knapsack Problem (MCKP) è una generalizzazione dell'ordinario problema del **0/1 Knapsack**, un famoso problema di ottimizzazione combinatoria. In particolare, nel Knapsack problem, è dato uno zaino (knapsack) che possa sopportare un determinato peso W. Inoltre sono dati N oggetti, ognuno dei quali caratterizzato da un peso e un profitto. Il problema si propone di scegliere quali di questi oggetti mettere nello zaino per ottenere il maggiore profitto senza eccedere il peso sostenibile dallo zaino stesso.

Nel MCKP l'insieme degli elementi è suddiviso in classi. La scelta binaria (binary choice) di inserire un oggetto all'interno del knapsack è sostituita dalla selezione di un elemento per ciascuna classe di oggetti.

Inoltre é importante definire che, questi sono problemi computazionalmente difficili, infatti essi rientrano nella categoria dei problemi **NP-Complete**. Il che implica che sono "*problemi non deterministici in tempo polinomiale*" e quando é possibile calcolare una soluzione, non é detto che questa sia ottima.

1.1 Formulazione Matematica

Indichiamo con:

- n: numero di oggetti
- p_j: profitto associato all'oggetto j
- w_j: peso associato all'oggetto j
- W: capacità totale del knapsack.
- m: numero di classi in cui sono partizionati i diversi oggetti. $C_1 \cup C_2 \cup \cdots \cup C_m = \{1, 2, \ldots, \mathrm{n}\}$. C_r rappresenta l'intersezione delle diverse classi e C_s rappresenta l'insieme vuoto.
- $x_j \in \{0,1\}$: $\begin{cases} x_j = 1, \text{ se l'oggetto i è inserito nel knapsack} \\ x_j = 0, \text{ altrimenti} \end{cases}$

`FORMULAZIONE MATEMATICA`

$$
(\mathrm{P}): \begin{cases} max & \displaystyle\sum_{j=1}^{n} p_j x_j & & (1) \\ \text{s.t.} & \displaystyle\sum_{j=1}^{n} w_j x_j \leq W & & (2) \\ & \displaystyle\sum_{j \in C_k} x_j \leq 1 & \mathrm{k} = 1, \ldots, \mathrm{m} & (3) \\ & x_j \in \{0,1\} & & (4) \end{cases}
$$

Capitolo 2

Risoluzione del problema

Per risolvere il MCKP si è deciso di adottare la programmazione dinamica.

2.1 Programmazione Dinamica

La programmazione dinamica è una tecnica di risoluzione dei problemi, introdotta negli anni '40 dal matematico statunitense *Richard Bellman*.

Sostanzialmente la programmazione dinamica è un metodo per risolvere un problema complesso, suddividendolo in più sottoproblemi più semplici, risolvendo ciascuno di questi una sola volta e memorizzando le loro soluzioni utilizzando una struttura di dati basata sulla memoria (array, mappa, ecc.). Ciascuna delle soluzioni dei sottoproblemi viene indicizzata in qualche modo, in genere in base ai valori dei relativi parametri di input, in modo da facilitare la sua ricerca. Quindi la volta successiva che si verifica lo stesso sottoproblema, invece di ricalcolare la sua soluzione, si può semplicemente cercare la soluzione precedentemente calcolata, risparmiando così tempo di calcolo. Questa tecnica di memorizzazione di soluzioni per sottoproblemi invece di ricalcolarle è chiamata **memoization**.

Costo risoluzione = O($\sum n$ x W)

2.2 Esempio

Partendo dalla problema (P), abbiamo che:

$$
\begin{aligned}
max\ z = & \overbrace{x_1 \quad x_2 \quad x_3}^{J_1=\{1,3\}} \overbrace{x_4 \quad x_5 \quad x_6 \quad x_7}^{J_2=\{4,7\}} \overbrace{x_8 \quad x_9 \quad x_{10}}^{J_3=\{8,10\}} \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} && \leq W \\
& x_1 + x_2 + x_3 && \leq 1 \\
& x_4 + x_5 + x_6 + x_7 && \leq 1 \\
& x_8 + x_9 + x_{10} && \leq 1
\end{aligned}
$$

2.3 Sottoproblema

Ponendo:

- $J_r,\ dove\ r = 1, \ldots, m$ l'insieme che denota gli oggetti appartenenti ad una determinata classe r.
- x^* la soluzione ottima corrispondente alle prime k classi ($k < n$). Ovvero:

$$x^*_{J_1}, \ldots, x^*_{J_2-1} x^*_{J_2}, \ldots, x^*_{J_k}, \ldots, x^*_{J_{k+1}-1}$$

  Quindi x^* é la soluzione ottima del problema $f(q, k)$ sotto riportato.

- $q = \sum_{J=1}^{J_{k+1}-1} w_j$

FORMULAZIONE MATEMATICA

$$f(q,k): \begin{cases} max & \sum_{J=1}^{J_{k+1}-1} p_j x_j & (5) \\ \text{s.t.} & \sum_{J=1}^{J_{k+1}-1} w_j x_j = q & (6) \\ & \sum_{J=J_r}^{J_{k+1}-1} x_j \leq 1 \qquad \text{r} = 1, \ldots, \text{k} & (7) \\ & x_j \in \{0,1\} & (8) \end{cases}$$

2.4 Ricursione di programmazione dinamica

- Nessun oggetto della classe k é in soluzione:

$$f(q,k) = f(q,k-1)$$

- Altrimenti:

$$f(q,k) = \max_{\substack{J_k \leq J \leq J_{k+1} \\ s.t.\ q-w_J \geq 0}} \{f(q-w_J, k-1) + p_J\}$$

Definito ciò, la ricursione di programmazione dinamica risulta essere:

$$f(q,k) = \max\{f(q,k-1), \max_{\substack{J_k \leq J \leq J_{k+1} \\ s.t.\ q-w_J \geq 0}} \{f(q-w_J, k-1) + p_J\}\}$$

Dove:

- $k = 1, \ldots, m$
- $q = 1, \ldots, W$

2.5 Soluzione

In questa sessione si espongono le scelte implementative per la risoluzione del MCKP.

2.5.1 Algoritmo

1. Inizializzazione:
   - $f(q,0) = 0$ per $q = 0, \ldots, W$
   - $f(0,k) = 0$ per $k = 1, \ldots, m$
2. Per ogni $q = 0, \ldots, W$ e per ogni $k = 1, \ldots, m$ tale per cui $f(q,k) \geq 0$ calcola:

$$f(q,k) = \max\{f(q,k-1), \max_{\substack{J_k \leq J \leq J_{k+1} \\ s.t.\ q-w_J \geq 0}} \{f(q-w_J, k-1) + p_J\}\}$$

   - Se nessun oggetto della classe k é in soluzione:

$$f(q,k) = f(q,k-1)$$

   - Altrimenti:

$$f(q,k) = \max_{\substack{J_k \le J \le J_{k+1} \\ s.t.\ q - w_J \ge 0}} \{f(q - w_J, k-1) + p_J\}$$

3. Se $k = J_{k+1}$, ovvero è l'ultima classe e $q = n$, ovvero si sono processati tutti gli elementi della classe k: **STOP**.

   Altrimenti: poni $k = k + 1$ e ritorna allo step (2)

2.6 Implementazione

Per l'implementazione del MCKP, si è deciso di usare il *C++* è un linguaggio di programmazione ad alto livello, orientato agli oggetti, con tipizzazione statica. In questa accezione si è pensato di sfruttare un approccio più rapido e intuitivo.

Come si può notare in Figure 2.1, si é deciso di spezzare la complessità implementativa suddividendo il carico di lavoro. In particolare si può trovare:

- **OneOrNoneForClass**: permette di computare il MCKP nella variante in cui un oggetto i appartenente alla classe k sia scelto o meno per l'inserimento all'interno del knapsack.
- **MultipleChoiceKnapsackProblem**: cuore della computazione di programmazione dinamica che si occupa di computare la ricursione.
- **PrintMatrix**: oggetto di sostegno utile per la stampa dei risultati.
- **ComputeSolution**: si occupa di predisporre l'input.

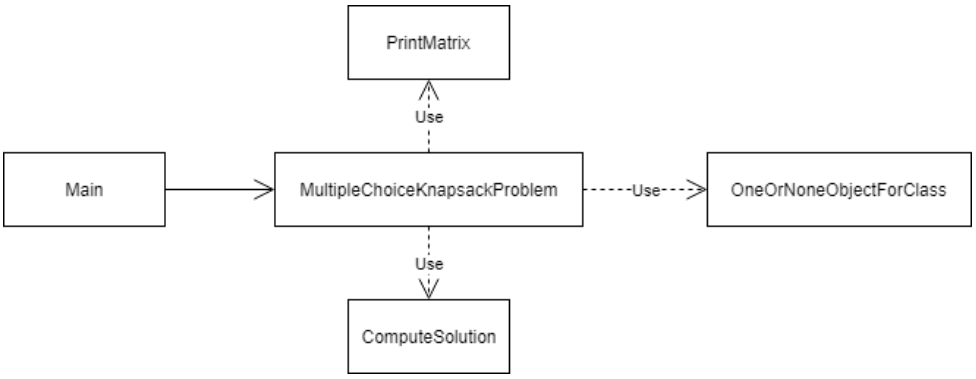

Figura 2.1: General Domain Model

2.7 Esempio

Di seguito verrà descritto un semplice esempio.

2.7.1 Parametri in Input

- Class 1:

Oggetti	x_1	x_2	x_3	x_4
Profitto	15	11	5	8
Peso	8	4	4	3

- Class 2:

Oggetti	x_5	x_6	x_7	x_8
Profitto	12	18	20	14
Peso	5	14	11	5

- Capacità knapsack: 10

2.8 Output

- Tabella ricursione:

q — k	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	8	8	8	8	8
4	0	11	5	8	11	11	11	11
5	0	11	5	8	12	11	11	14
6	0	11	5	8	12	11	11	14
7	0	11	5	8	12	11	11	14
8	15	11	5	8	20	15	15	22
9	15	11	5	8	23	15	15	25
10	15	11	5	8	23	15	15	25

- Oggetti inseriti nel knapsack:
  - Classe 1: oggetto x_2
  - Classe 2: oggetto x_8
- Risultati:
  - Profitto totale: 25
  - Peso totale: 9

Capitolo 3

Varianti

Poniamo ora alcune varianti del MCKP.

3.1 Scelta di esattamente un elemento per ogni classe

$$\text{(P')}: \begin{cases} max & \sum_{j=1}^{n} p_j x_j & & (1) \\ \text{s.t.} & \sum_{j=1}^{n} w_j x_j \leq W & & (2) \\ & \sum_{j \in C_k} x_j = 1 & \text{k} = 1, \ldots, \text{m} & (3') \\ & x_j \in \{0,1\} & & (4) \end{cases}$$

In questo caso, il problema prevede che per ogni classe, deve essere obbligatoriamente scelto *esattamente* uno ed un solo oggetto da inserire all'interno del knapsack. Infatti il vincolo (3) del problema (P) diviene un vincolo di uguaglianza (3') nel problema (P').

3.1.1 Ricursione di programmazione dinamica

- Nessun oggetto della classe k é in soluzione: non può verificarsi.
- Altrimenti:

$$f(q,k) = \max_{\substack{J_k \leq J \leq J_{k+1} \\ s.t.\ q - w_J \geq 0}} \{f(q - w_J, k-1) + p_J\}$$

Quindi banalmente la ricursione di programmazione dinamica diviene:

$$f(q,k) = \max_{\substack{J_k \leq J \leq J_{k+1} \\ s.t.\ q - w_J \geq 0}} \{f(q - w_J, k-1) + p_J\}\}$$

Dove:

- $k = 1, \ldots, m$
- $q = 1, \ldots, W$