Linguaggi di Programmazione: progetto "The Knapsack Problem"

A.A. 2011-2012

Il problema

Il Knapsack Problem (KP), o problema dello Zaino, è un classico problema di ottimizzazione combinatoria. E' dato un insieme I di oggetti ed uno zaino; ad ogni oggetto $i \in I$ è associato un peso w_i ed un valore p_i; allo zaino è associata una capacità C. Supponiamo che i dati del problema siano interi non negativi. L'obiettivo è scegliere un sottoinsieme di oggetti $\bar{I} \subseteq I$ la cui somma dei pesi non ecceda la capacità dello zaino

$$\sum_{i \in \bar{I}} w_i \leq C$$

e la cui somma dei valori sia massima

$$\bar{I} \in \text{argmax}_{\tilde{I} \subseteq I} \{ \sum_{i \in \tilde{I}} p_i \}.$$

Non sono ammessi frazionamenti: ogni oggetto deve essere inserito nello zaino per intero, oppure scartato. Può essere considerato il problema di ottimizzazione combinatoria più semplice, dato che richiede di ottimizzare una funzione obiettivo lineare e la soluzione è soggetta ad un singolo vincolo, esprimibile come una disuguaglianza lineare. Le condizioni di integralità sulla scelta degli oggetti, tuttavia, lo rendono un problema NP-Hard [1]: non sono noti algoritmi risolutivi con costo in tempo polinomiale.

Un algoritmo

Trattandosi di un problema fondamentale, che sorge sia indipendentemente che come parte di scenari più complessi, il KP è molto studiato. In letteratura sono stati proposti diversi algoritmi per ottenere delle soluzioni esatte [2]. Un famoso algoritmo risolutivo, che sfrutta tecniche di programmazione dinamica [3], è il seguente.

Senza perdita di generalità, supponiamo che gli oggetti nell'insieme I siano ordinati secondo un criterio arbitrario, indichiamo con $n = |I|$ il numero di oggetti, e supponiamo quindi l'insieme I possa essere rappresentato come $I = \{1, 2, \ldots, n\}$. Indichiamo con f_{kc} il *valore* della *miglior combinazione di oggetti* che

(a) include solo oggetti di indice compreso tra 1 e k;

(b) ha somma dei pesi pari a c.

Il valore

$$\max_{0 \leq c \leq C} \{f_{nc}\}$$

corrisponde alla soluzione ottima del problema (può includere qualsiasi oggetto ed ammette una somma dei pesi qualsiasi che non ecceda la capacità dello zaino).

Intuitivamente, la miglior soluzione ad un problema che include solo oggetti di indice compreso tra 1 e k ed utilizza al massimo c unità di capacità può essere ottenuta analizzando due soli casi:

- **non includendo** l'oggetto k-esimo, ed in tal caso la miglior soluzione è la stessa che si ottiene considerando solo oggetti di indice compreso tra 1 e $k - 1$;
- **includendo** l'oggetto k-esimo, ed in tal caso la miglior soluzione è si ottiene utilizzando w_k unità di capacità per il nuovo oggetto, e sfruttando nel modo migliore le rimanenti $c - w_k$ unità di capacità per gli oggetti precedenti.

Quindi, per ogni $k \in \{0 \dots n\}$ e per i valori di capacità utilizzata $c \in \{0 \dots C\}$, i valori f_{kc} possono essere calcolati in modo ricorsivo come segue:

$$f_{kc} = \begin{cases} 0 & \text{se } k \leq 0 \\ f_{k-1,c} & \text{se } k > 0 \text{ e } w_k > c \\ \max\{f_{k-1,c}, p_k + f_{k-1,c-w_k}\} & \text{altrimenti.} \end{cases}$$

La valutazione di tale funzione ricorsiva può avvenire per via diretta, ed in tal caso il relativo costo in tempo è esponenziale, oppure sfruttando tecniche di memoization tipo accumulatori (come visto a lezione per i numeri di Fibonacci o per il calcolo del fattoriale), ed in tal caso tutti i valori f_{kc} possono essere calcolati in tempo $O(nC)$ (pseudo-lineare).

Parallelizzazione

Il calcolo dei valori f_{kc} si presta bene ad un approccio concorrente.

Concorrenza a livello di statement: fissato un valore di k, tutti i termini f_{kc} possono essere calcolati in modo indipendente. Quindi un approccio concorrente consiste nel calcolo dei valori f_{kc} "riga per riga": si parte con $k = 0$ ed in parallelo si calcolano i valori f_{0c} per $0 \leq c \leq C$, si passa a $k = 1$ calcolando i valori f_{1c} in parallelo e via dicendo.

Concorrenza a livello di sottoprogrammi: ogni valore di f_{kc} può essere calcolato tramite l'attivazione di un task indipendente; tali task devono essere sincronizzati, permettendo l'esecuzione del task per f_{kc} solo quando i task per $f_{k-1,c}$ e $f_{k-1,c-w_k}$ hanno terminato la propria esecuzione.

Riferimenti bibliografici

[1] M.R. Garey, D.S. Johnson (1979), Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman.

[2] S. Martello, P. Toth (1990) Knapsack problems – algorithms and computer implementations, John Wiley and sons.

[3] A.V. Aho, J.E. Hopcroft, J.D. Ullman (1983) Data Structures and Algorithms, Addison-Wesley.