Scheduling single-machine: disegno, implementazione e test di uno schema di branch-and-bound combinatorio

Carmine Scarpitta
email
matricola

Elly Schmidt
email
matricola

Davide Romano Tranzocchi
email
matricola

A.A. 2017/2018

Indice

1 Introduzione

In questo documento saranno presentati gli aspetti salienti dell'applicazione sviluppata per risolvere il problema di scheduling $1|r_j|\sum_j C_j$ impiegando un algoritmo di branch-and-bound combinatorio.

2 Caratteristiche teoriche e idee algoritmiche implementate

2.1 Algoritmo di branch-and-bound

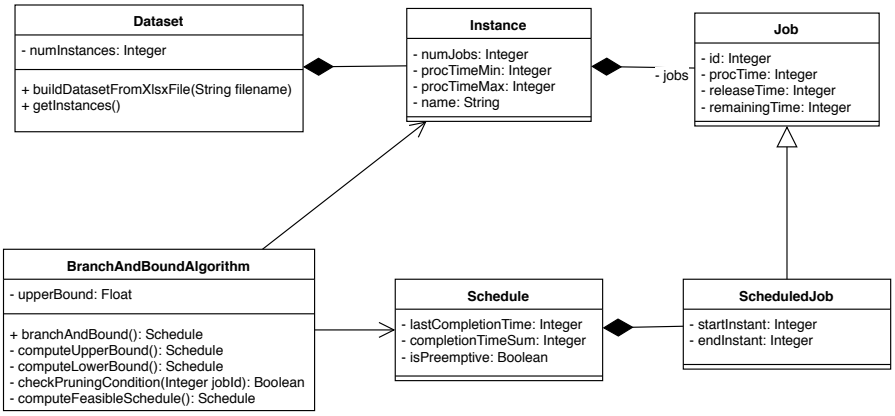
L'algoritmo di branch-and-bound da noi implementato si articola in tre fasi:

1. Inizializzazione
   (a) Calcola un upper bound sul valore della funzione obiettivo da una schedula ammissibile
   (b) Inizializza sequenze parziali 'partialSequence' e 'remainingJobs'
   (c) Inizialmente poni tutti i nodi cioè i job in 'remainingJobs'
2. Primo passo di ramificazione
   (a) FOR Nodo = 1 .. n DO
      i. Seleziona il nodo j-esimo secondo un criterio e aggiungilo a 'partialSequence'
      ii. Calcola un lower bound sulla soluzione ottima per quel nodo calcolando una schedula con prelazione
      iii. Se $LB < UB$ continue
3. Passi successivi di ramificazione

3 Documentazione del codice

L'applicazione dapprima legge un file Excel contenente le istanze del problema di scheduling considerato impiegando le API fornite dal componente XSSF del progetto Apache POI. I moduli di questo componente consentono di creare, modificare, leggere e scrivere fogli elettronici nel formato di file .xlsx. Dunque l'applicazione costruisce un dataset a partire dalle istanze tramite un parser che analizza ogni cella del foglio elettronico e crea nuovi oggetti Java della classe Job inizializzandoli con i valori dei tempi di processamento e di rilascio contenuti all'interno del file. Infine applica l'algoritmo di branch-and-bound a ciascuna istanza al fine di ottenere la soluzione del problema di scheduling corrispondente.

4 Schema delle classi



5 Risultati sperimentali

Presentiamo ora i risultati dei test svolti sulle varie istanze, considerando prima le soluzioni dell'implementazione in Java e poi in AMPL.

5.1 Schema implementato in Java

- Istanza 1: $\sum_j C_j = 3520$, schedula ottima $\sigma = (8, 2, 6, 7, 5, 1, 10, 3, 9, 4)$
- Istanza 2: $\sum_j C_j = 15177$, schedula ottima $\sigma = (29, 9, 14, 16, 25, 7, 12, 17, 8, 28, 19, 27, 3, 11, 26, 30, 24, 20,$
- Istanza 3: $\sum_j C_j = 9694$, schedula ottima $\sigma = (7, 19, 23, 16, 4, 12, 28, 21, 9, 26, 11, 3, 5, 17, 30, 6, 8, 10, 24,$
- Istanza 4: $\sum_j C_j = 32688$, schedula ottima $\sigma = (35, 14, 1, 33, 8, 38, 40, 6, 30, 23, 11, 32, 13, 21, 12, 17, 2, 28,$
- Istanza 5: $\sum_j C_j = 16236$, schedula ottima $\sigma = (3, 30, 11, 25, 38, 13, 31, 26, 39, 14, 4, 20, 12, 24, 27, 37, 23,$
- Istanza 6: $\sum_j C_j = 87651$, schedula ottima $\sigma = (40, 8, 36, 30, 3, 51, 33, 43, 20, 52, 35, 5, 53, 16, 44, 4, 41, 56,$