Progetto
Algoritmi e modelli per l'ottimizzazione discreta

Scarpitta	Schmidt	Tranzocchi
email	email	email
matricola	matricola	matricola

A.A. 2017/2018

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. Il progetto considerato è il numero 3 della tipologia A.

2 Installazione e avvio

Per poter eseguire il codice Java del progetto è necessario avere installato Java JDK oppure Java JRE, mentre per eseguire il codice AMPL è necessario l'omonimo programma. Una volta ottenuta la cartella contenente il progetto, per eseguire l'applicazione Java basta aprire la cartella `amod-scheduling/src/` e avviare la classe `Main.java` con un ambiente di sviluppo integrato oppure direttamente da un terminale dopo aver compilato il codice sorgente. Invece per eseguire il codice AMPL è necessario avviare l'interprete dei comandi AMPL oppure l'interfaccia grafica AMPLIDE, spostarsi nella cartella `amod-progetto/ampl/` tramite il comando `cd` e lanciare il comando AMPL `include scheduling.run`. Per decidere quale istanza del problema di scheduling testare basta modificare la linea di codice `read table jobs1;` e sostituire il numero 1 con il numero dell'istanza desiderata.

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 Risultati sperimentali

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

4.1 Schema implementato in Java

- Istanza 1: $\sum_j C_j = 3520$
- Istanza 2: $\sum_j C_j = 15177$
- Istanza 3: $\sum_j C_j = 9694$
- Istanza 4: $\sum_j C_j = 32688$
- Istanza 5: $\sum_j C_j = 16236$

- Istanza 6: $\sum_j C_j = 87651$

4.2 Schema implementato in AMPL

Test effettuati su un calcolatore dotato di processore Intel Core i3 di terza generazione.

- Istanza 1: $\sum_j C_j = 3479$
- Istanza 2: $\sum_j C_j = 16047$
- Istanza 3: $\sum_j C_j = 9780$
- Istanza 4: $\sum_j C_j = 34009$
- Istanza 5: $\sum_j C_j = 16954$
- Istanza 6: $\sum_j C_j = 87948$

5 Analisi sperimentale

Dai test si evince che i valori delle soluzioni fornite dalle due implementazioni sono confrontabili. Si ipotizza che lasciando proseguire la computazione per lo schema AMPL sia possibile ottenere soluzioni ottime migliori.

Riferimenti bibliografici

[1] A. Pacifici. *Note del corso di Algoritmi e modelli per l'ottimizzazione discreta*. 2018.

[2] A. Agnetis. *Dispensa "Introduzione ai problemi di scheduling"*. 2000.

[3] R. Fourer, D. M. Gay, B. W. Kernighan. *"Database access"*, Capitolo 13 di "AMPL: A Modeling Language for Mathematical Programming". Seconda edizione.