UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Triennale in Ingegneria Informatica

Implementazione MATLAB e test di problemi di ottimizzazione lineare con obiettivi multipli e priorità

Relatore:

Prof. Marco Cococcioni

Candidato:

Gregorio Maria Manduzio

ANNO ACCADEMICO 2023/2024

Indice

1 Introduzione

In matematica, l'ottimizzazione consiste nel trovare la migliore soluzione, scelta in base ad uno o più criteri, ad un problema. Tale soluzione, se esiste, dovrà essere ricercata all'interno dell'insieme delle soluzioni ammissibili di quel problema.

1.1 Classificazione dei problemi di ottimizzazione

Esistono numerosi tipi distinti di problemi di ottimizzazione; tuttavia, alcuni di essi presentano caratteristiche simili e possono essere raggruppati in classi, sotto-classi. Una prima grande classificazione è la distinzione tra problemi di Programmazione Lineare (PL), Programmazione Lineare Intera (PLI) e Programmazione Non Lineare (PNL). Il problema trattato in questa tesi, cosiddetto *makespan scheduling*, appartiene alla seconda tra queste categorie, la PLI, ed in particolare alla sotto-categoria dei problemi di assegnamento. Esso verrà approfondito nel capitolo 2.

1.2 Risoluzione dei problemi di ottimizzazione

È facile accorgersi di come l'ottimizzazione sia un concetto generale ed applicabile ad una vastissima gamma di situazioni reali. In particolare, risulta profondamente legato, tra gli altri, ai concetti di efficienza e di produttività. Si pensi ad un piano di strategia aziendale: un aumento dei profitti si potrà perseguire attraverso un aumento dei ricavi oppure attraverso una diminuzione dei costi di produzione, ottenibile ad esempio gestendo opportunamente i periodi di lavoro dei macchinari.

Tali problemi possono assumere dimensioni importanti e necessitano di software *ad hoc* per essere risolti. Le licenze d'uso di tali software, tra i quali si possono trovare CPLEX e Gurobi, hanno spesso dei prezzi elevati. Uno degli intenti di questa tesi è la realizzazione di un risolutore che possa essere eseguito da un programma che, pur non essendo a sua volta gratuito, è più economico e certamente più diffuso: MATLAB.

2 Il problema del *makespan scheduling*

In questo capitolo, ad una breve introduzione riguardante gli aspetti fondamentali della PLI farà seguito la presentazione del problema analizzato in questa tesi.

2.1 Programmazione Lineare Intera

Un problema di Programmazione Lineare Intera, nella sua forma standard, si presenta così:

$$\begin{cases} \min c^T \cdot x & (1) \\ A \cdot x \leq b & (2) \\ x \in \mathbb{Z}^n & (3) \end{cases}$$

dove x è il vettore soluzione.

Il rigo (1) del sistema rappresenta la volontà di minimizzare una certa **funzione obiettivo** la quale, essendo lineare, può essere rappresentata attraverso un vettore (in questo caso c), moltiplicato scalarmente per il vettore soluzione x. La disequazione (2) impone quelli che sono i **vincoli** del problema; è di nuovo possibile, trattandosi di funzioni esclusivamente lineari, condensare tutti i vincoli in una matrice A e richiedere che il prodotto tra quest'ultima ed il vettore soluzione sia minore o uguale ad un certo vettore b. Infine l'ultimo vincolo, (3), marca la sostanziale differenza tra i problemi di PL ed i problemi di PLI: il vettore soluzione deve essere a componenti intere.

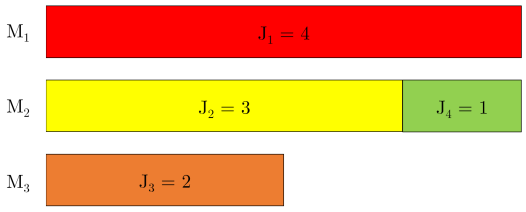
Bisogna specificare che questo sistema rappresenta soltanto il formato standard dei problemi di PLI. Niente vieta di massimizzare una funzione obiettivo invece di minimizzarla, oppure di impostare dei vincoli nella forma $A \cdot x \geq b$; l'interezza delle componenti del vettore soluzione rimane ad ogni modo fondamentale.

2.2 Minimizzazione del tempo di completamento su macchinari identici

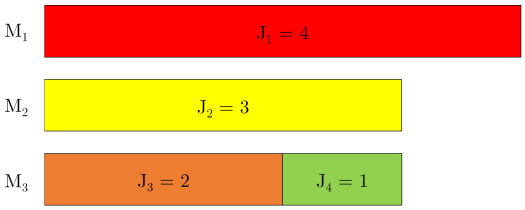
Si supponga di avere a disposizione m macchinari identici e capaci di lavorare in parallelo; non è importante specificare il tipo di macchinario - potrebbe anche trattarsi, ad esempio, dei *core* di un processore. Un numero n di lavori - o processi, nel caso dei *core* - devono essere eseguiti dagli m macchinari. Ciascun lavoro ha un suo tempo di completamento e deve essere eseguito per intero, ossia senza poter essere interrotto e successivamente ripreso. Chiaramente, essendo i macchinari identici, il tempo di completamento è un parametro intrinseco del lavoro e assolutamente indipendente dal macchinario che viene utilizzato per eseguirlo. L'obiettivo è assegnare i lavori ai macchinari in modo tale che l'ultimo macchinario a completare il proprio set di lavori finisca il prima possibile.

2.2.1 Esempio

Si prenda in considerazione il caso in cui $m = 3$ ed il set è composto da $n = 4$ lavori della durata di 1, 2, 3 e 4 minuti. La soluzione ottima di questo problema è ovvia e vede il lavoro da 4 minuti "dominare" sugli altri ed essere assegnato in solitaria ad un macchinario. I restanti lavori potranno essere assegnati agli altri macchinari per formare così le due combinazioni {4; (3, 1); 2} oppure {4; 3; (2, 1)}.



primo possibile assegnamento



secondo possibile assegnamento

Risulta ovvio come, sempre a causa dell'uguaglianza tra macchinari, una qualsiasi permutazione di macchinari all'interno di uno stesso assegnamento ottimo non alteri la soluzione; i macchinari sono numerati e le soluzioni ordinate in modo decrescente esclusivamente per una questione di chiarezza. La presenza di più soluzioni ottime è di vitale importanza per questa tesi e deve pertanto essere tenuta in grande considerazione.

2.2.2 Descrizione formale del problema - caso mono-obiettivo

L'articolo [1] presenta il sistema che descrive interamente il problema, qui riportato:

$$
\begin{cases}
\min C_{\max} & & \\
C_{\max} \geq C_i & M_i \in \mathcal{M} & (1) \\
C_i = \sum_{j=1}^{n} x_{i,j} \cdot p_j & M_i \in \mathcal{M} & (2) \\
\sum_{i=1}^{m} x_{i,j} = 1 & J_j \in \mathcal{J} & (3) \\
x_{i,j} \in \{0,1\} & J_j \in \mathcal{J}, M_i \in \mathcal{M} & (4)
\end{cases}
$$

dove M_i rappresenta l'i-esimo macchinario appartenente al set $\mathcal{M}$ di tutti i macchinari, J_j rappresenta il j-esimo lavoro appartenente al set $\mathcal{J}$ di tutti i lavori (d'ora in avanti chiamati "*job*") e p_j la sua durata.

I vincoli raggruppati in (1) rappresentano l'espressione $C_{max} = \max_{1 \leq i \leq m} C_i$, ossia il massimo *makespan*, ciò che si vuole minimizzare, è il maggiore tra i *makespan* di tutti i macchinari. I vincoli (2) assicurano che tutti i macchinari, in ogni istante, eseguano al più un *job*. I vincoli (3) impongono che ogni *job* sia eseguito esattamente da un macchinario. Il vincolo (4) esprime la natura di questo problema, un assegnamento binario; in particolare $x_{i,j}$ vale 1 se il *job* j è assegnato al macchinario i e 0 altrimenti.

3 L'approccio lessicografico

Fino a questo momento si è considerato il caso "classico" di minimizzazione di una singola funzione obiettivo. Cosa succederebbe se, invece, ve ne fosse più d'una? Ecco che entrano in gioco i problemi lessicografici.

3.1 Caso generale

Si fornisce di seguito un esempio, assolutamente non correlato al *makespan scheduling* bensì più "pratico" e di più semplice comprensione, di problema lessicografico.

Una persona è interessata all'acquisto di una nuova automobile. Questa persona, per compiere la scelta migliore, valuta tre parametri, qui elencati in ordine di priorità decrescente: il prezzo, il consumo di carburante ed il comfort. Dopo aver eliminato le automobili troppo costose, troppo poco efficienti o troppo poco confortevoli, restano le possibilità elencate nella tabella sottostante.

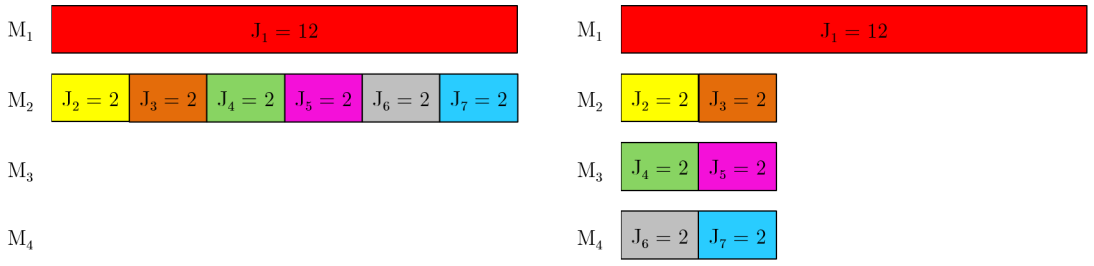
	Prezzo	Consumo	Comfort
Automobile 1	25.000 €	7 $l/100km$	9/10
Automobile 2	25.000 €	10 $l/100km$	7/10
Automobile 3	28.000 €	8 $l/100km$	10/10
Automobile 4	25.000 €	7 $l/100km$	8/10

La persona scarterà per prima l'automobile 3 in quanto più costosa delle altre, poi scarterà l'automobile 2 poiché, a parità di prezzo, è quella con i consumi peggiori, infine sceglierà l'automobile 1 dato che, a parità di prezzo e consumi, risulta essere la più confortevole.

Questo piccolo esempio lascia intendere come si possa risolvere un problema lessicografico: si minimizza/massimizza la funzione obiettivo con massima priorità, subito dopo la seconda più importante (scelta tra le soluzioni "ugualmente ottime" ottenute al passo precedente) e così via. È necessario che ad ogni passo si generi un set di soluzioni ottime sufficientemente ampio per avere un discreto margine di manovra ai successivi.

3.2 Applicazione al problema del *makespan scheduling*

Risolvere il problema del *makespan scheduling* mono-obiettivo può, in alcuni casi, portare a soluzioni piuttosto "sbilanciate". Per comprendere meglio questo concetto, si prendano in considerazione le due figure seguenti.



È evidente come le due soluzioni siano ugualmente ottime, tuttavia si tende a preferire l'assegnamento in figura (b) a quello in figura (a) proprio perché più "bilanciato".

Questa scelta viene fatta per riuscire a contrastare eventuali malfunzionamenti dei macchinari. Si supponga di optare per l'assegnamento (a) e, dato che rimarrebbero inutilmente attivi, di spegnere i macchinari M_3 ed M_4. Dopo l'esecuzione del *job* J_4 si verifica un guasto al macchinario M_2; i *job* non possono essere riassegnati e la soluzione ottima, nel migliore dei casi, peggiora di un tempo pari al tempo necessario per riparare il macchinario. Se invece si optasse per l'assegnamento (b), supposto un guasto a M_2 dopo l'esecuzione di J_2, si potrebbe riparare il macchinario senza probabilmente peggiorare la soluzione ottima; in alternativa, accodare il *job* J_3 ad uno tra M_3 e M_4 sicuramente non altererebbe il massimo tempo di completamento.

3.2.1 Descrizione formale del problema - caso lessicografico

Il seguente lemma, tratto dall'appendice [2] e qui riportato con dimostrazione, consente di formulare il sistema che descrive il problema del *makespan scheduling* nel caso lessicografico.

Lemma 1. *In una soluzione ottima al problema del makespan scheduling lessicografico:*

1. $C_i \geq C_{i+1}$ $\qquad \forall i = 1, \ldots, m-1$
2. $i \cdot C_i + \sum_{q=i+1}^{m} C_q \leq \sum_{j=1}^{n} p_j \leq \sum_{q=1}^{i-1} C_q + (m-i+1) \cdot C_i$ $\qquad \forall i = 1, \ldots, m$

Dimostrazione. In ogni assegnamento ammissibile i macchinari possono essere riordinati per soddisfare la prima proprietà. Per quanto riguarda la seconda, si noti come $\sum_{i=1}^{m} C_i = \sum_{j=1}^{n} p_j$. Dal momento che $C_i \geq \ldots \geq C_m$ si ottiene che $\sum_{q=1}^{i-1} C_q + (m-i+1) \cdot C_i \geq \sum_{j=1}^{n} p_j$. Analogamente, dato che $C_1 \geq \ldots \geq C_i$ si conclude che $i \cdot C_i + \sum_{q=i+1}^{m} C_q \leq \sum_{j=1}^{n} p_j$. □

Questo, invece, è il sistema precedentemente menzionato [1]:

$$
\begin{cases}
\text{lexmin } C_1, \ldots, C_m & \\
C_i \geq C_{i+1} & M_i \in \mathcal{M} \setminus M_m \\
\displaystyle\sum_{q=1}^{i-1} C_q + (m - i + 1) \cdot C_i \geq \sum_{j=1}^{n} p_j & M_i \in \mathcal{M} \\
\displaystyle i \cdot C_i + \sum_{q=i+1}^{m} C_q \leq \sum_{j=1}^{n} p_j & M_i \in \mathcal{M} \\
\displaystyle C_i = \sum_{j=1}^{n} x_{i,j} \cdot p_j & M_i \in \mathcal{M} \\
\displaystyle\sum_{i=1}^{m} x_{i,j} = 1 & J_j \in \mathcal{J} \\
x_{i,j} \in \{0, 1\} & J_j \in \mathcal{J}, M_i \in \mathcal{M}
\end{cases}
$$

4 Realizzazione di script MATLAB per la risoluzione del problema

Analogamente a quanto visto nel capitolo 2 si presentano, dopo una breve introduzione all'ambiente MATLAB, gli script necessari per la risoluzione del problema del *makespan scheduling*, sia nel caso mono-obiettivo che in quello lessicografico. Fanno seguito dei commenti per agevolare la comprensione del codice ed una valutazione dei risultati ottenuti.

4.1 L'ambiente MATLAB

MATLAB è una piattaforma di programmazione e calcolo numerico per l'analisi di dati, lo sviluppo di algoritmi e la creazione di modelli. All'avvio l'applicazione mostra un'interfaccia a riga di comando simile, per certi versi, ad un comune terminale. Nella maggior parte dei casi l'utente impartisce comandi testuali (più precisamente alfanumerici) in input e riceve in output, sempre in forma testuale, le risposte. Negli ambiti della simulazione o della *data science* si può lavorare con strumenti più complessi, installati attraverso appositi *toolbox*, che ad esempio possono fornire risposte in forma grafica; nel caso degli script presentati in questa tesi l'input-output testuale è sufficientemente espressivo.

Come un qualsiasi linguaggio di programmazione MATLAB fornisce all'utente la possibilità di creare e salvare su file degli script che realizzano funzioni personalizzate; questa è la funzionalità che ha consentito la realizzazione dei file **makespan_singolo.m** e **makespan_lessicografico.m**.

4.1.1 Il comando *intlinprog*

L'*Optimization Toolbox*, come suggerisce il nome, estende le funzionalità di MATLAB aggiungendo dei risolutori per problemi di ottimizzazione. Tra i comandi accessibili dopo l'installazione si ha *intlinprog*, ossia il risolutore per problemi di PLI. Esso risolve il problema

$$\begin{cases} \min c^T \cdot x \\ A \cdot x \leq b \\ Aeq \cdot x = beq & (1) \\ lb \leq x \leq ub & (2) \\ x(intcon) \in \mathbb{Z} \end{cases}$$

in cui si possono notare due vincoli in più rispetto al problema standard; si possono infatti, nell'equazione (1), specificare dei vincoli di uguaglianza, mentre il vincolo (2) imposta un limite inferiore e superiore per le componenti del vettore soluzione.

La sintassi MATLAB "classica" del comando è

```
x = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);
```

dove

- x è il vettore soluzione
- f è il vettore che rappresenta la funzione obiettivo
- *intcon* è un vettore che specifica quali componenti di x sono soggette al vincolo di interezza (si noti come nel sistema appena presentato si ha $x(intcon) \in \mathbb{Z}$ e non $x \in \mathbb{Z}^n$)
- i restanti parametri sono quelli già nominati nel sistema

Il numero di parametri, tanto in input quanto in output, è variabile. In input, a seconda delle proprie esigenze, alcuni di questi possono essere omessi, oppure ne possono essere aggiunti per specificare ad esempio le condizioni di stop del *branch-and-bound*, l'algoritmo utilizzato da *intlinprog* per risolvere il problema. È infine possibile ottenere risultati più dettagliati e statistiche sull'esecuzione di quest'ultimo aggiungendo parametri in output [1].

4.2 makespan_singolo.m

```
1  function [] = makespan_singolo(macchinari, tempi)
2
3  % variabili
4
5  jobs = length(tempi);
6  dim_soluzione = (macchinari * jobs) + macchinari + 1;
7
8  % costruzione vettore c
9
10 c = zeros(dim_soluzione, 1);
11 c(1) = 1;
12
13 % vincoli di interezza
14
15 intcon = 1:dim_soluzione;
16
17 % costruzione matrice A
18
19 A = zeros(macchinari, dim_soluzione);
20 A(1:macchinari, 1) = -1;
21 A(1:macchinari, 2:(macchinari + 1)) = eye(macchinari);
22
23 % costruzione vettore b
24
25 b = zeros(1, macchinari);
26
27 % costruzione matrice Aeq
```

[1] maggiori informazioni sono reperibili all'indirizzo `https://it.mathworks.com/help/optim/ug/intlinprog.html`

```
28
29 Aeq = zeros((macchinari + jobs), dim_soluzione);
30 Aeq(1:macchinari, 2:(macchinari + 1)) = eye(macchinari);
31
32 for i = 1:macchinari
33     Aeq(i, ((jobs * (i - 1)) + macchinari + 2):((jobs * i) + macchinari...
34     + 1)) = (tempi * -1);
35 end
36
37 for i = 1:jobs
38     for j = (macchinari + i + 1):jobs:((jobs * (macchinari - 1))...
39         + macchinari + i + 1)
40
41         Aeq((macchinari + i), j) = 1;
42
43     end
44 end
45
46 % costruzione vettore beq
47
48 beq = [zeros(1, macchinari), ones(1, jobs)];
49
50 % definizione dei bound
51
52 lb = zeros(dim_soluzione, 1);
53 ub = [Inf((macchinari + 1), 1); ones((macchinari * jobs), 1)];
54
55 % opzioni da passare ad intlinprog
56
57 options = optimoptions('intlinprog', 'Display', 'off');
58
59 % chiamata ad intlinprog
60
61 assegnamento = intlinprog(c, intcon, A, b, Aeq, beq, lb, ub, options);
62
63 % visualizzazione risultati
64
65 risultati = transpose(reshape(assegnamento((macchinari + 2):...
66             dim_soluzione), jobs, macchinari));
67
68 nomi_righe = {};
69 for i = 1:macchinari
70     nomi_righe = [nomi_righe, {strcat('Macchinario', num2str(i))}];
71 end
72
73 nomi_colonne = {};
74 for i = 1:jobs
75     nomi_colonne = [nomi_colonne, {strcat('Job', num2str(i))}];
76 end
77
78 risultati = array2table(risultati, 'VariableNames', nomi_colonne,...
79             'RowNames', nomi_righe);
80 fprintf('\nMAKESPAN\n\n');
81
82 for i = 2:(macchinari + 1)
83     fprintf('\tC%u = %u\n', (i-1), assegnamento(i));
84 end
85
86 fprintf('\nASSEGNAMENTO\n\n');
87 disp(risultati);
88 fprintf('SOLUZIONE\n\n\t%u\n\n', assegnamento(1));
```

4.3 makespan_lessicografico.m

```
function [] = makespan_lessicografico(macchinari, tempi)

% variabili

jobs = length(tempi);
dim_soluzione = (macchinari * jobs) + macchinari + 1;

% costruzione vettore c

c = zeros(dim_soluzione, 1);
c(1) = 1;

% vincoli di interezza

intcon = 1:dim_soluzione;

% costruzione matrice A

A = zeros(macchinari, dim_soluzione);
A(1:macchinari, 1) = -1;
A(1:macchinari, 2:(macchinari + 1)) = eye(macchinari);

% costruzione vettore b

b = zeros(1, macchinari);

% costruzione matrice Aeq

Aeq = zeros((macchinari + jobs), dim_soluzione);
Aeq(1:macchinari, 2:(macchinari + 1)) = eye(macchinari);

for i = 1:macchinari
    Aeq(i, ((jobs * (i - 1)) + macchinari + 2):((jobs * i) + macchinari...
    + 1)) = (tempi * -1);
end

for i = 1:jobs
    for j = (macchinari + i + 1):jobs:((jobs * (macchinari - 1))...
        + macchinari + i + 1)

        Aeq((macchinari + i), j) = 1;

    end
end

% costruzione vettore beq

beq = [zeros(1, macchinari), ones(1, jobs)];

% definizione dei bound

lb = zeros(dim_soluzione, 1);
ub = [Inf((macchinari + 1), 1); ones((macchinari * jobs), 1)];

% opzioni da passare ad intlinprog

options = optimoptions("intlinprog", "Display", "off");

% ciclo di chiamate ad intlinprog

for i = 1:macchinari

```

```
63      if i == 1
64          assegnamento = intlinprog(c, intcon, A, b, Aeq, beq, lb,...
65          ub, options);
66      else
67          assegnamento = intlinprog(c, intcon, A, b, Aeq, beq, lb,...
68          ub, assegnamento, options);
69      end
70
71      if i == macchinari
72          break;
73      end
74
75      % aggiunta vincolo Ci = <tempo di completamento appena trovato>
76
77      [nuovo_min, indice] = max(assegnamento((1 + i):(1 + macchinari)));
78      nuovo_vincolo = zeros(1, dim_soluzione);
79      nuovo_vincolo(i + 1) = 1;
80      Aeq = [Aeq; nuovo_vincolo];
81      beq = [beq, nuovo_min];
82
83      % rimozione del primo vincolo da A
84
85      A(1, :) = [];
86      b(1) = [];
87
88      % eventuale scambio negli assegnamenti per rendere ammissibile la
            soluzione di partenza
89
90      if indice > 1
91          indice = indice + i - 1;
92
93          assegnamento([(i + 1), (indice + 1)]) =...
94          assegnamento([(indice + 1), (i + 1)]);
95
96          assegnamento([(((i - 1) * jobs) + macchinari + 2):((i * jobs) +...
97          macchinari + 1), (((indice - 1) * jobs) + macchinari + 2):...
98          ((indice * jobs) + macchinari + 1)]) = assegnamento...
99          ([(((indice - 1) * jobs) + macchinari + 2):((indice * jobs)...
100         + macchinari + 1), (((i - 1) * jobs) + macchinari + 2):...
101         ((i * jobs) + macchinari + 1)]);
102     end
103 end
104
105 % visualizzazione risultati
106
107 risultati = transpose(reshape(assegnamento((macchinari + 2):...
108             dim_soluzione), jobs, macchinari));
109
110 nomi_righe = {};
111 for i = 1:macchinari
112     nomi_righe = [nomi_righe, {strcat('Macchinario', num2str(i))}];
113 end
114
115 nomi_colonne = {};
116 for i = 1:jobs
117     nomi_colonne = [nomi_colonne, {strcat('Job', num2str(i))}];
118 end
119
120 risultati = array2table(risultati, 'VariableNames', nomi_colonne,...
121             'RowNames', nomi_righe);
122 fprintf('\nMAKESPAN\n\n');
123
124 for i = 2:(macchinari + 1)
125     fprintf('\tC%u = %u\n', (i-1), assegnamento(i));
```

```
126 end
127
128 fprintf('\nASSEGNAMENTO\n\n');
129 disp(risultati);
130 fprintf('SOLUZIONE\n\n\t%u\n\n', max(assegnamento(2:(macchinari + 1))));
```

4.4 Commenti al codice

I due script presentati mostrano diverse analogie ma anche alcune differenze. Come si può notare la prima sezione di entrambi (righe 1-54) è dedicata alla preparazione degli input da passare ad *intlinprog* ed è del tutto identica. La visualizzazione in forma matriciale di tali input si trova nella sezione successiva. Le due variabili dichiarate all'inizio del codice vengono utilizzate per evitare di dover eseguire più volte lo stesso calcolo; la seconda, in particolare, indica la dimensione del vettore soluzione che contiene, in ordine, il *makespan* massimo, tutti i *makespan* (compreso quest'ultimo) ed infine tutte le variabili che rappresentano l'assegnamento macchinario-*job*. Allo stesso modo anche la parte finale, dedicata alla visualizzazione dei risultati, non presenta alcuna differenza. Essi vengono visualizzati nel seguente ordine: prima i *makespan* di tutti i macchinari, poi l'assegnamento dei *job* (in forma tabellare) ed infine la soluzione ottima del problema.

La differenza risiede nella parte centrale del codice, in cui viene invocata la funzione *intlinprog*: mentre nel caso mono-obiettivo una singola chiamata è sufficiente per risolvere il problema, se si vuole adoperare un approccio lessicografico ciò non basta. Il metodo sequenziale implementato prevede che la funzione venga chiamata un numero di volte pari al numero di macchinari di cui si dispone. Lo pseudo-codice che descrive l'algoritmo è il seguente [2]:

1: $v_1^* = \min\{C_1 : (\vec{x}, \vec{C}) \in \mathcal{S}\}$
2: **for** $i = 2, \ldots, m$ **do**
3: $\quad v_i^* = \min\{C_i : x \in \mathcal{S}, C_1 = v_1^*, \ldots, C_{i-1} = v_{i-1}^*\}$
4: Restituisci il risultato ottenuto all'ultima iterazione

dove $\vec{x}$, $\vec{C}$ e $\mathcal{S}$ rappresentano, rispettivamente, il vettore degli assegnamenti macchinario-*job*, il vettore dei *makespan* ed il set di soluzioni ammissibili.

In pratica al passo 1 si risolve il problema come se fosse mono-obiettivo, mentre nelle successive iterazioni si richiede che i precedenti *makespan* siano uguali ai risultati ottenuti nelle iterazioni passate.

4.4.1 *Warm start*

Una particolarità dello script **makespan_lessicografico.m** si può apprezzare alle righe 63-69 e 90-102.

Nelle prime si nota come, a partire dalla seconda iterazione, si fornisca ad *intlinprog* l'assegnamento ottenuto all'iterazione precedente. Ciò viene fatto affinché il

branch-and-bound abbia un punto di partenza, cosa che rende l'esecuzione immensamente più veloce. Questo miglioramento è dovuto al fatto che il vincolo aggiunto dopo ogni iterazione è piuttosto "stringente"; fornendo al risolutore un assegnamento già ammissibile si evita di dover ricalcolare tutti gli assegnamenti che danno origine precisamente al *makespan* appena ottenuto.

A causa dell'ordinamento decrescente dei *makespan* è possibile - anzi, probabile - che l'assegnamento appena ottenuto non sia ammissibile e sia dunque inaccettabile come punto di partenza. Questa situazione si verifica quando all'i-esima iterazione il *makespan* minimo non risulta appartenente all'i-esimo bensì al k-esimo, $k > i$, macchinario. In tal caso le righe 90-102 si occupano di scambiare i set di lavori assegnati ai due macchinari, operazione assolutamente lecita in virtù della loro uguaglianza.

Per dare un'idea dell'importanza di questo accorgimento si è voluto comparare il tempo di risoluzione della prima istanza **degenerate/moderate** [3] nei due casi. L'elaboratore utilizzato monta una CPU AMD Ryzen 7 3800X, una memoria RAM da 16 GB e fa uso di Windows 11 Pro a 64 bit e della versione R2023b Update 7 di MATLAB. La versione presentata dello script ha risolto il problema in 0.329 secondi, mentre la versione senza il *warm start* e le operazioni di scambio ne ha impiegati ben 264.

4.5 Visualizzazione di matrici e vettori

Per una maggiore comprensione si presentano le matrici ed i vettori effettivamente utilizzati nel processo di risoluzione del problema, in entrambe le sue versioni. Si prenda a modello, per semplicità, un problema con $m = 3$, $n = 5$ ed i tempi di completamento [1, 2, 3, 4, 5]. Si utilizzi come riferimento, per il seguente caso mono-obiettivo, il sistema contenuto nella sotto-sezione 2.2.2.

$$c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

Il vettore c, in questa forma, specifica che si vuole minimizzare la prima variabile, ossia C_{max}

$$A = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

A e b rappresentano i vincoli raggruppati in (1).

$$A_{eq} = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & -2 & -3 & -4 & -5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & -3 & -4 & -5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & -3 & -4 & -5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$b_{eq} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Le prime m righe della matrice A_{eq}, e di conseguenza le prime m componenti del vettore b_{eq}, realizzano i vincoli (2), mentre il resto rappresenta i vincoli (3).

$$intcon = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \end{bmatrix}$$

$$lb = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$$ub = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$$

Questa combinazione di *intcon*, *lb* e *ub* esprime il vincolo (4).

Si esaminano ora le matrici ed i vettori dati in input ad *intlinprog* nel caso lessicografico. Alla prima iterazione sono identici, come detto, al caso mono-obiettivo; A, b, A_{eq} e b_{eq}, d'altro canto, vengono modificati dopo ogni iterazione - fatta eccezione, ovviamente, per l'ultima - per essere coerenti con quanto scritto nei commenti inseriti nello script. Ad esempio, tra la prima e la seconda iterazione si impostano

$$A = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

per rimuovere il vincolo $C_{max} \geq C_1$ e

$$A_{eq} = \begin{bmatrix} 0 & 1 & 0 & 0 & -1 & -2 & -3 & -4 & -5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & -3 & -4 & -5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -2 & -3 & -4 & -5 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$b_{eq} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & v_1^* \end{bmatrix}$$

per aggiungere il vincolo $C_1 = v_1^*$.

4.6 Valutazione dei risultati ottenuti

Le istanze di problemi del *makespan scheduling* possono essere suddivise in **bilanciate** e **multimodali**. Come suggerisce il nome, dalle prime si ottengono con una singola iterazione dei *makespan* tutti uguali o comunque molto simili tra loro; viceversa le istanze multimodali, pur restituendo una soluzione ottima, non tengono conto di questo aspetto. Di conseguenza, volendo comparare i risultati prodotti dai due script, bisogna utilizzare istanze del secondo tipo. Queste ultime sono costituite da un numero arbitrario di macchinari ed un vettore di tempi di completamento dei *job* in cui la quasi totalità delle componenti fanno parte di un intorno di raggio sufficientemente piccolo, mentre le pochissime altre assumono valori più elevati. Un esempio è il problema presentato nella sezione 2 del terzo capitolo.

Effettuando dei test con istanze di questo tipo e di dimensioni ristrette si è giunti alla conclusione che effettivamente lo script lessicografico restituisce soluzioni più bilanciate e "stabili": spesso, fornendo allo script mono-obiettivo una semplice permutazione del vettore dei tempi di completamento, si ottengono *makespan* diversi. Preso atto di questa differenza, si è voluto testare lo script lessicografico con istanze di dimensioni maggiori la cui soluzione è nota ed ottenuta mediante il software CPLEX [3]. I *makespan* ottenuti, pur non essendo identici, sono assai simili; in alcuni test risultano addirittura più bilanciati. Prese le prime 10 istanze della categoria **degenerate/moderate**, infatti, se si calcola

$$\frac{(C_{max}^{MATLAB} - C_{min}^{MATLAB}) - (C_{max}^{CPLEX} - C_{min}^{CPLEX})}{C_{max}^{CPLEX} - C_{min}^{CPLEX}} \cdot 100$$

si ottiene uno "sbilanciamento" medio del 2.73%.

Riferimenti bibliografici

[1] Dimitrios Letsios, Miten Mistry, and Ruth Misener, 2020, *Exact Lexicographic Scheduling and Approximate Rescheduling*, European Journal of Operational Research, Volume 290, Issue 2, 469-478, `https://www.sciencedirect.com/science/article/pii/S0377221720307451`

[2] Dimitrios Letsios, Miten Mistry, and Ruth Misener, 2020, *Supplementary Material for "Exact Lexicographic Scheduling and Approximate Rescheduling"*, `https://www.sciencedirect.com/science/article/pii/S0377221720307451#sec0019`

[3] Dimitrios Letsios, 2018, *Exact lexicographic scheduling methods and approximate recovery strategies for two-stage makespan scheduling*, `https://github.com/cog-imperial/two_stage_scheduling`

Letture addizionali

[4] Patrick Halder, Fabian Christ, and Matthias Althoff, 2023, *Lexicographic Mixed-Integer Motion Planning with STL constraints*, in Proc. of the 26th Int. Conf. on Intelligent Transportation Systems.

[5] Gabriele Eichfelder, Tobias Gerlach, and Leo Warnow, 2023 *A test instance generator for multiobjective mixed-integer optimization*, Mathematical Methods of Operations Research, 1-26