Politecnico di Torino

Report Homework 2 Business Analytics

Lorenzo Leoni (s291305), Giuseppe Biagio Lapadula (s292220)

5 Agosto 2021

Abstract

Il problema che si sta trattando è un caso di Lot Sizing Stocastico a capacità ridotta. Il nostro obbiettivo è risolverlo utilizzando la programmazione dinamica stocastica ed esatta (non è un approccio approssimato). Le simulazioni sono state effettuate in Matlab. Per questo problema abbiamo sviluppato un algoritmo basato sul modello descritto successivamente. In totale sono stati creati tre script:

- *MakePolicy*: script principale, che permette di trovare le politiche ottimali per ogni instante di tempo t ed ogni stato possibile;
- *SimulatePolicy*: script che permette di effettuare una simulazione MonteCarlo out of sample delle nostre politiche. Serve per effettuare la validazione successivamente nello script *ProvaMakePolicy*.
- *ProvaMakePolicy*: script dove si inseriscono le variabili, si runnano gli algoritmi e si effettuano validazione e calcolo di errori vari;

Per testarli è necessario inserirli tutti nella stessa cartella.

Modello Matematico

In questa sezione andremo a esporre il problema che si sta trattando, ovvero un caso di Lot Sizing Stocastico a capacità ridotta. Il nostro compito è gestire una linea di produzione che può lavorare 4 tipi di prodotti diversi e ne può lavorare uno alla volta in ogni turno di lavorazione, che chiameremo *"Time Bucket" Tb*. Il nostro obiettivo è organizzare la linea in modo tale da soddisfare la domanda per i 4 tipi di prodotto e contemporaneamente minimizzare il costo totale, che dipende da vari parametri:

- *"Costo di giacenza" h*: esso dipende dallo stato del magazzino (numero di items al suo interno) all'inizio di un Time Bucket.
- *"Costo di setup" F*: è il costo del passaggio della produzione da un prodotto ad un altro.
- *"Costo di penalità" w*: in questo caso si assume la domanda "lost sales", cioè l'invenduto è perso, causandoci una perdita quantificata dal nostro costo w.

Per quanto riguarda la produzione, decidiamo di produrre una certa quantità fissata di prodotto p per ogni item. Inoltre, quando si passa da un prodotto ad un altro (ovvero si fa un setup), si impiega un certo tempo u dove non si può produrre l'item (si può avere un tempo u diverso per ogni tipo di prodotto), ed esso è supposto essere $u \leq Tb$, dove Tb è la durata del *time bucket*. Nel restante tempo residuo post-setup si produce una frazione del totale p. Assumiamo la domanda discreta e stocastica, dipendente da una distribuzione di probabilità assegnata. La nostra linea di produzione lavorerà per T+1 time buckets.

Da qui possiamo tirar fuori il seguente problema di ottimizzazione:

$$\min_{x_{i,t},s_{ji,t}} \left\{ E_0 \left[\sum_{t=0}^{T} \left(\sum_{i=1}^{4} \left(h_i I_{i,t} + \sum_{j=1}^{4} F_{ij} s_{ji,t} + w_i \cdot \max_{x_{i,t},s_{ji,t}} \left\{ d_{i,t+1} - p_i x_{i,t} \left(1 - \frac{u_i}{Tb} \cdot \mathbb{1}_{\left\{ \sum_{j=1}^{4} s_{ij,t} \right\}} \right), 0 \right\} \right) \right) \right] \right\}$$

Con vincoli

$$I_{i,t+1} = \min_{x_{i,t},s_{ji,t}} \left\{ 0, I_{i,t} + p_i x_{i,t} \left(1 - \frac{u_i}{Tb} \cdot \mathbb{1}_{\left\{ \sum_{j=1}^{4} s_{ij,t} \right\}} \right) - d_{i,t+1} \right\}$$

$$\sum_{i=1}^{4} x_{i,t} \leq 1$$

$$s_{ji,t} \geq x_{i,t} - x_{i,t-1}, \quad \text{per una } j \neq i$$

$$\sum_{j=1}^{4} s_{ji,t} \leq 1$$

$$x_{it} \in \{0, 1\}$$

$$s_{ji,t} \in \{0, 1\}$$

$$I_{i,t} \leq Imax_i$$

Dove $x_{i,t}$ indica la decisione se produrre o meno il prodotto i nel time bucket t, $s_{ji,t}$ indica la decisione se effettuare il setup dal prodotto j al prodotto i, al tempo t ed $I_{i,t}$ indica lo stato del magazzino per il prodotto i al tempo t. $d_{i,t}$ è una variabile aleatoria che indica l'ammontare della domanda per il prodotto i nell'intervallo t (cioè l'intervallo fra gli istanti $t-1$ e t).

Questo problema è approcciabile sfruttando la programmazione dinamica, grazie alla struttura multistadio dovuta alla sua suddivisione in time bucket, in cui possiamo generare una struttura ricorsiva. Per fare ciò sono state effettuate delle modifiche alle variabili e ai termini che calcolano la funzione obbiettivo:

- Quello che si fa inizialmente è creare uno spazio degli stati: il nostro stato sarà un vettore 5-dimensionale, di cui 4 dimensioni indicano la quantità per ogni prodotto nei 4 spazi di magazzino rispettivi, mentre la quinta è un valore che indica la decisione presa al tempo precedente $t-1$, che chiameremo λ_t. Ciò funziona perchè dato il limitato numero di decisioni che si possono prendere, è possibile enumerarle facilmente.

- Per quanto riguarda le decisioni, la variabile $s_{ji,t}$ diventa $s_{i,t}$, indicandoci solo il prodotto su cui andiamo ad effettuare un setup. Inoltre viene aggiunta anche un'altra decisione l_t, che indica quale prodotto stiamo producendo, tale che

  $$l_t = i \ se \ x_{i,t} = 1$$

  oppure

  $$l_t = i \ se \ s_{i,t} = 1$$

  oppure

  $$l_t = i \ se \ x_{j,t} = 0 \ e \ s_{i,t} = 0, \ i = 1...4 \ e \ l_{t-1} = i$$
  $$i = 1, 2, 3, 4$$

  In sostanza, rispettando i vincoli, notiamo che si hanno tre tipi di decisione, in base ai valori assunti:

  1. Decisione di sola produzione per il prodotto i, in cui si ha

     $$x_{i,t} = 1, \ x_{j,t} = 0, \ j \neq i$$

     $$s_{j,t} = 0, \ i, j \in \{1, 2, 3, 4\}$$

     ,

     $$l_t = i$$

  2. Decisione di setup(+ produzione parziale), tale che:

     $$x_{i,t} = 1, \ s_{i,t} = 1$$

     $$x_{j,t} = 0, \ s_{j,t} = 0, \ j \neq i \ l_t = i$$
     $$i, j \in \{1, 2, 3, 4\}$$

3. Decisione di stop produzione, tale che:

$$x_{i,t} = 0,\ s_{i,t} = 0 i = 1, 2, 3, 4,$$

$$l_t = l_{t-1}$$

- Inoltre va fatta una modifica alla funzione obbiettivo nel termine che calcola il costo di setup, date le modifiche alla variabile: F è una matrice 4×4 a valori non negativi, tale che la sua diagonale è formata da 0. Riscrivo il calcolo come:

$$F_{l_{t-1},l_t} \cdot \max_i(s_{i,t})$$

Date queste informazioni, si può procedere a scrivere l'equazione di ottimalità di bellman in modo da ottenere una *Value Function*, che da il meccanismo iterativo alla risoluzione del problema:

$$V_t(I_t, \lambda_t) = \min_{x_t, s_t, l_t} \Bigg\{ \sum_{i=1}^{4} h_i \cdot I_{it} + F_{l_{t-1},l_t} \cdot \max_i(s_{i,t}) \ + \\ + \ E\Bigg[\sum_{i=1}^{4} \Big(- \Big(\min_{x_t, s_t} \{ I_{i,t} + p_i x_{i,t} \big(1 - \frac{u_i}{Tb}\big) \cdot \max_i(s_{i,t}) - d_{t+1}, 0 \} \Big) + V_{t+1}\Big(I_{t+1} | I_t, x_t, s_t, l_t \Big) \Bigg] \Bigg\}$$

Si hanno le seguenti transizioni di stato:

$$I_{i,t+1} = \min_{x_{i,t}, s_{ji,t}} \Big\{ 0, I_{i,t} + p_i x_{i,t} \Big(1 - \frac{u_i}{Tb} \cdot \max_i(s_{i,t}) - d_{i,t+1} \Big\}$$

$$\lambda_{t+1} = [x_t, s_t, l_t]^T$$

Per quanto riguarda le variabili di stato e decisionali abbiamo:

- x_t : vettore le cui componenti sono $x_{i,t}$;
- s_t : vettore le cui componenti sono $s_{i,t}$;
- I_t : vettore le cui componenti sono $I_{i,t}$;
- $l_{t-1} = \lambda_9$, dove la componente 9 è quella corrispondente ad l_t.

Alcuni dettagli algoritmo MakePolicy

In *MakePolicy* definiamo una function per poter individuare le politiche ottimali, la quale è in funzione di diversi parametri tutti definiti nelle righe iniziali del codice in forma di commento. Definiamo l'actionTable e la valueTable in forma matriciale e tensoriale, quest'ultima è fondamentale per poterci permettere di svolgere i calcoli all'interno dell'algoritmo in modo più chiaro. Definiamo x come la matrice di tutte le azioni (in totale 12) che possono essere compiute, dove le prime 4 componenti ci indicano se stiamo producendo un prodotto (quindi $\exists!\ i = 1, ..., 4 \mid x_i = 1$) e di quale prodotto stiamo parlando (corrispondente all'indice i). La componente 5 fino alla componente 8 ci indicano se per produrre l'item i abbiamo dovuto attuare un setup, mentre la componente 9 ci indica su che item è configurata la macchina. Nella matrice feasible la riga i si riferisce all'azione della colonna i-esima della matrice x, indicando se da essa può essere compiuta o meno l'azione j, ovvero se nella colonna j-esima della matrice feasible è presente 1 o 0 rispettivamente, corrispondente all'azione della colonna j-esima della matrice x. Successivamente si procede con il salvare tutte le possbili combinazioni di magazzino e dell'azione sulla quale ci baseremo per poter sceglierne una feasible e che sia la migliore possibile. Da qui in poi inizia l'algoritmo vero e proprio, il quale è ben commentato nello script Matlab *MakePolicy.m*.

Simulazioni

Validazione

Per verificare la bontà dell'algoritmo *MakePolicy* abbiamo effettuato delle simulazioni Montecarlo out of sample.

Fissati i seguenti parametri:

$$Imax = [5, 5, 5, 5]^T$$
$$demandProbs = \frac{[1, 1, 1, 1, 1, 1]}{6}$$
$$p = [1, 2, 3, 4]^T$$
$$u = \frac{[1, 1, 1, 1]^T}{2}$$
$$h = [1, 2, 3, 4]^T$$
$$w = [2, 4, 6, 8]^T$$

$$\begin{cases} F = 5 \cdot mean(h) \cdot \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

$$Tb = 1$$

N.B.: Abbiamo utilizzato questa struttura matriciale poichè rispetta bene l'idea dei prodotti divisi in due famiglie, cioè la famiglia $\{1, 2\}$ e la famiglia $\{3, 4\}$, tali che il prezzo per passare da un prodotto ad un altro della stessa famiglia sia minore di quello per passare ad un prodotto di una famiglia diversa. Ovviamente, per fare ciò si può utilizzare qualsiasi altra configurazione matriciale.

Il primo test è fare una simulazione, su un numero di scenari da generare pari a 1000 e fissando lo stato iniziale $[0, 0, 0, 0, 1]^T$. Otteniamo:

Costo dato dalla ValueTable (istante iniziale) = 193.1466

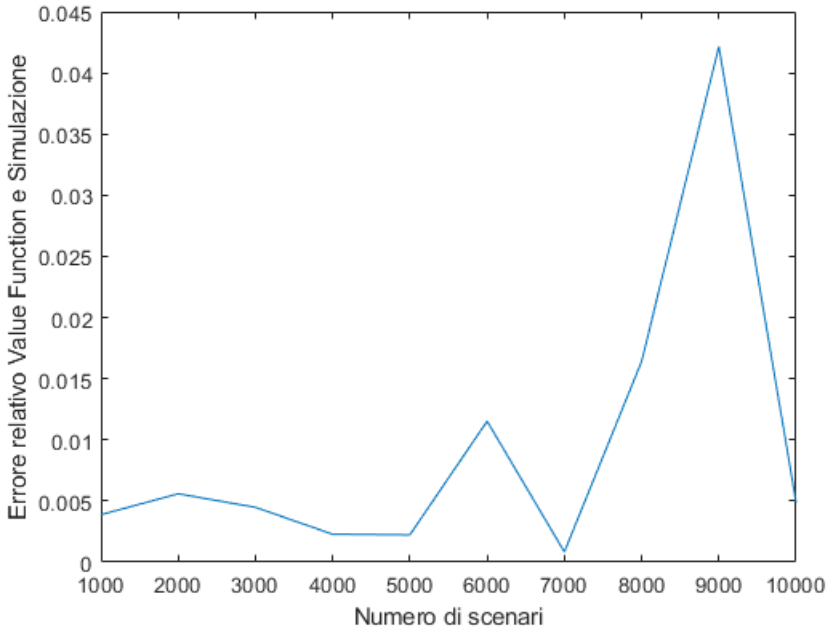
Costo dato dalle simulazioni = 193.1010

Si nota subito che i due risultati sono molto vicini, ciò fa quindi pensare che l'algoritmo funzioni correttamente con questa configurazione.

Il secondo test ci permette di testare la robustezza dell'algoritmo variando il numero di scenari su cui si effettuano le simulazioni, facendolo ogni volta per uno stato di partenza casuale, così da poter comprendere se ci sono forti scostamenti o meno dell'errore di calcolo. Le simulazioni sono state fatte con un numero di scenari che varia tra 1000 e 10000, variando di volta in volta di 1000 scenari.

Nell'esecuzione di questo controllo abbiamo ottenuto il seguente grafico:

Numero Scenari VS Errore Relativo



Dal grafico si può notare che le fluttuazioni del nostro errore viaggiano al di sotto dell'ordine di 10^{-2}, quindi possiamo considerarlo accettabile, concludendo che il nostro algoritmo sia robusto.

Nell'ultimo test effettuato abbiamo considerato tutti gli stati iniziali possibili, fissando il numero di scenari pari a 100 considerandone quindi i rispettivi costi per poter calcolare varie misure d'errore per poter avere una conoscenza generale sul problema:

Errore in norma infinito = 4.6034

Errore in norma infinito su minimo valore della Value function = 0.0333

Errore relativo medio = 0.0279

Gli errori relativi hanno dei valori dell'ordine di 10^{-2}, che possono essere considerati accettabili nel nostro caso: il primo infatti ci limita superiormente l'errore relativo che le simulazioni possono generare, mentre il secondo ci da una stima dei valori che l'errore può assumere.

N.B.: I valori degli errori possono variare leggermente a causa della randomicità dei test.

Risultati

N.B.: Per quanto riguarda la lettura dell'action table, l'output è una matrice con numero di righe quanto il numero totale di stati e numero di colonne $T + 5$ (dove T è l'orizzonte temporale). Nelle ultime 5 colonne della matrice i numeri codificano le varie combinazioni di stato $[I_1, I_2, I_3, I_4, l_{t-1}]$, mentre nelle altre vi sono dei valori che ci indicano qual è l'azione ottimale. In particolare (ogni tipo di azione in ordine oer prodotti 1,2,3,4):

- Azioni dalla 1 alla 4: azioni di produzione.
- Azioni dalla 5 alla 8: azioni di setup.
- Azioni dalla 9 alla 12: azioni di stato idle (si differenziano poiché indicano su quale pezzo è montata la linea di produzione).

Sono stati effettuati diversi test riguardanti i vari parametri dell'algoritmo, variando:

1. Dimensione spazio degli stati
2. Durata time bucket VS durata setup
3. Domanda rispetto alla capienza del magazzino e variabilità della domanda
4. Costo di setup VS costo di giacenza
5. Impatto penalità
6. Impatto capacità produttiva

1) Dimensione spazio degli stati

Il nostro spazio degli stati è dato dalle possibili configurazioni dei magazzini $I_i, i = 1, ..., 4$. Per come è strutturato l'algoritmo *MakePolicy* ci aspettiamo che con l'aumentare delle dimensioni delle capacità di essi il tempo computazionale aumenterà a sua volta.

Fissati i seguenti parametri:

$$demandProbs = \frac{[1, 1, 1, 1, 1]}{5}$$

$$p = [2, 4, 3, 2]^T$$

$$u = \frac{[1,1,1,1]^T}{2}$$
$$h = [3,4,5,2]^T$$
$$w = [1,1,1,1]^T$$

$$\begin{cases} F = \frac{mean(h)}{10} . * \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$u = \frac{[1,1,1,1]^T}{2}$$
$$T = 4$$
$$Tb = 1$$

Di seguito riportiamo i risultati ottenuti per alcune capacità massime di magazzino:

Imax	Tempo (s)
$[5,5,5,5]$	11.956468
$[10,10,10,10]$	128.946919
$[15,15,15,15]$	610.480388
$[20,5,5,5]$	36.280092

Come avevamo premesso, con l'aumentare della capacità massima dei magazzini avremmo ottenuto un maggior tempo computazionale dell'algoritmo *MakePolicy*.

2) Durata time bucket VS durata setup

Analizziamo l'impatto del rapporto tra la durata del setup e del time bucket. Decidiamo prima di fissare il vettore del setup u e variare il time bucket Tb, poi faremo il contrario. Chiaramente ciò che verrà influenzato dalla variazione di questo rapporto non è il tempo computazionale, (il quale dipende dal numero di configurazioni di magazzino e delle azioni), ma semplicemente i calcoli e i risultati delle nostre simulazioni. Quindi ciò che andremo a valutare è il valore della Value Function nei diversi casi, fissando lo stato iniziale $[0,0,0,0,1]$.

Fissati i seguenti parametri:

$$Imax = [10, 10, 10, 10]^T$$

$$demandProbs = \frac{[1, 1, 1, 1, 1, 1]}{6}$$

$$p = [5, 5, 5, 5]^T$$

$$h = [3, 4, 5, 2]^T$$

$$w = [1, 1, 1, 1]^T$$

$$\begin{cases} F = \frac{mean(h)}{10} . * \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

Di seguito riportiamo i risultati ottenuti fissando il costo di setup $u = [1, 1, 1, 1]$:

Tb	Costo
1	37.5000
2	36.4792
3	37.1278
4	37.1278
5	37.5000

Di seguito riportiamo i risultati ottenuti fissando il costo di setup $Tb = 5$:

u	Costo
[2, 3, 1, 4]	37.5000
[5, 2, 5, 5]	37.5000
[5, 2, 2, 5]	37.5000

Notiamo che i costi rimangono simili, concludendo che il rapporto tra i due parametri non impatta molto sulla variazione del costo (Value Function).

Tb = 5, u = [2,2,2,2]

9	9	9	8	6	9	6	2	1
10	10	10	8	6	9	6	2	2
11	11	11	8	6	9	6	2	3
12	12	12	4	6	9	6	2	4
9	9	9	8	6	9	6	2	5
10	10	10	8	6	9	6	2	6
11	11	11	8	6	9	6	2	7
12	12	12	4	6	9	6	2	8
9	9	9	8	6	9	6	2	9
10	10	10	8	6	9	6	2	10
11	11	11	8	6	9	6	2	11
12	12	12	4	6	9	6	2	12
9	9	9	8	6	9	6	3	1
10	10	10	8	6	9	6	3	2

Tb = 5, u = [5,5,5,5]

9	9	9	9	6	9	6	2	1
10	10	10	10	6	9	6	2	2
11	11	11	11	6	9	6	2	3
12	12	12	4	6	9	6	2	4
9	9	9	9	6	9	6	2	5
10	10	10	10	6	9	6	2	6
11	11	11	11	6	9	6	2	7
12	12	12	4	6	9	6	2	8
9	9	9	9	6	9	6	2	9
10	10	10	10	6	9	6	2	10
11	11	11	11	6	9	6	2	11
12	12	12	4	6	9	6	2	12
9	9	9	9	6	9	6	3	1
10	10	10	10	6	9	6	3	2

3) Domanda VS capienza del magazzino e variabilità della domanda

Supponiamo di avere una domanda distribuita uniformemente, lo stato iniziale pari a $[0, 0, 0, 0, 1]$ e definiamo il parametro dm come la dimensione del vettore di domanda.

Fissati i seguenti parametri:

$$Imax = [5, 5, 5, 5]^T$$

$$demandProbs = \frac{ones(1, dm)}{dm}$$

$$p = [2, 4, 3, 2]^T$$

$$u = \frac{[1, 1, 1, 1]^T}{2}$$

$$h = [4, 4, 4, 4]^T$$

$$w = [4, 4, 4, 4]^T$$

$$\begin{cases} F = \frac{mean(h)}{10} . * \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

$$Tb = 1$$

Di seguito riportiamo i risultati ottenuti facendo variare il valore dm:

dm	Costo	Tempo (s)
2	29.3000	4.547964 s
6	137.8255	14.608906 s
10	261.4499	88.834938 s

I risultati sono in linea con quelli che ci aspettavamo. Infatti all'aumentare della dimensione della domanda aumenta il tempo computazionale, poichè nell'algoritmo *MakePolicy* procediamo con il calcolare tutti i possibili stati del magazzino al variare della domanda, maggiore è quest'ultima e maggiore saranno i calcoli da fare. Anche l'aumento del costo (Value Function) era atteso, poiché data la capacità che avevamo fissato all'inizio ($Imax = [5, 5, 5, 5]^T$) nel caso in cui $dm = 2$, ovvero si avrà la richiesta di 0 o 1 pezzo per ciascun item, sarà piu facile soddisfare la domanda. Al contrario nel caso in cui $dm = 10$, i pezzi richiesti variano da 0 a 9 per ogni item, quindi si avra una probabilità maggiore di non riuscire a soddisfare la domanda per almeno un item e quindi di incorrere in penalità.

Ora variamo la distribuzione della domanda. Di seguito riportiamo i risultati:

demandProbs	Costo
$[0.11, 0.11, 0.11, 0.22, 0.22, 0.23]$	167.3436
$[0.22, 0.22, 0.23, 0.11, 0.11, 0.11]$	109.8549
$[0, 0, 0, 1, 0, 0]$	160
$[0, 1, 0, 0, 0, 0]$	48.4000

Negli ultimi due casi rilassiamo il problema al caso deterministico, infatti ho informazione perfetta che la domanda assuma sempre valore fissato con probabilità 1.

4) Costo di setup VS costo di giacenza

Definiamo il rapporto tra costo di giacenza e costo di setup nel seguente modo: variamo k che serve per calcolare $\sum_{i=1}^{4} h_i \cdot \frac{1}{4} \cdot \frac{F_1}{k}$, con F_1 matrice che indica le proporzioni dei prezzi per setup da un tipo ad un altro (tramite la quale applichiamo il costo di setup major-minor per prodotti di famiglie diverse). Supponiamo di partire lo stato iniziale $[0, 0, 0, 0, 1]^T$.

Fissati i seguenti parametri:

$$Imax = [5, 5, 5, 5]^T$$

$$demandProbs = \frac{[1, 1, 1, 1, 1, 1]}{6}$$

$$p = [2, 4, 3, 2]^T$$

$$u = \frac{[1, 1, 1, 1]^T}{2}$$

$$h = [4, 4, 4, 4]^T$$

$$w = [4, 4, 4, 4]^T$$

$$\begin{cases} F = \sum_{i=1}^{4} h_i \cdot \frac{1}{4} \cdot \frac{1}{k} \cdot \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

$$Tb = 1$$

Di seguito riportiamo i risultati ottenuti per alcuni valori di k:

k	Costo
$\frac{1}{20}$	140.9815
$\frac{1}{10}$	140.9815
$\frac{1}{2}$	140.4481
5	138.0078
10	137.8255
20	137.7344

Si può quindi notare che all'aumentare di questo rapporto si può osservare una leggera diminuzione del costo totale, poichè il costo di setup tende a diminuire.

Se invece si prova a far variare contemporaneamente h e F in modo tale da avere:

$$h = k \cdot [1, 1, 1, 1]^T$$

$$F = \frac{1}{k} \cdot \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix}$$

Fissiamo $w = [2, 4, 6, 8]^T$ e $demandProbs = \frac{[1,1,1,1]}{4}$ e otteniamo:

k	Costo
$\frac{1}{20}$	109.2133
$\frac{1}{10}$	108.4055
$\frac{1}{2}$	98.2959
2	95.3108
5	97.1047
10	101.1900
20	109.7750

Otteniamo che se si aumenta k (quindi il costo di giacenza cresce mentre quello di setup decresce), abbiamo che per $k < 1$ il costo diminuisce, per poi aumentare quando $k > 1$. Ciò quindi fa pensare che le configurazioni migliori per ottenere costi minori sono tali che il rapporto tra costo di giacenza e costo di setup giacciono in un intorno di $k = 1$. Quello che si può intuire è che i costi dovrebbero essere bilanciati per avere il caso migliore.

5) Impatto Penalità

Analizziamo l'impatto della penalità sui costi e sulle azioni, eventualmente anche variando i costi di giacenza. Supponiamo di partire dallo stato iniziale $[0, 0, 0, 0, 1]^T$

Fissati i seguenti parametri:

$$Imax = [5, 5, 5, 5]^T$$

$$demandProbs = \frac{[1,1,1,1,1,1]}{6}$$

$$p = [2,4,3,2]^T$$

$$u = \frac{[1,1,1,1]^T}{2}$$

$$\begin{cases} k = 10 \\ F = \sum_{i=1}^{4} h_i \cdot \frac{1}{4} \cdot \frac{1}{k} \cdot \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

$$Tb = 1$$

Qui andremo a variare la penalità ed eventualmente il costo di giacenza:

- Fisso $w = [0,0,0,0]^T$ e otteniamo che l'algoritmo non produce, poichè non avendo penalità per domanda insoddisfatta i suoi costi dipendono solamente dalle azioni legate alla produzione, alla giacenza e al setup. Di conseguenza deciderà di non produrre per non incorrere in questi costi, come possiamo notare nella figura sottostante (le azioni corrispondenti allo stare fermi hanno indice da 9 a 12) e quindi il costo totale sarà 0.

w = [0,0,0,0]', penalità nulla

1	2	3	4	5	6	7	8	9
9	9	9	1	0	0	0	0	1
10	10	10	2	0	0	0	0	2
11	11	11	3	0	0	0	0	3
12	12	12	4	0	0	0	0	4
9	9	9	1	0	0	0	0	5
10	10	10	2	0	0	0	0	6
11	11	11	3	0	0	0	0	7
12	12	12	4	0	0	0	0	8
9	9	9	1	0	0	0	0	9
10	10	10	2	0	0	0	0	10
11	11	11	3	0	0	0	0	11
12	12	12	4	0	0	0	0	12
9	9	9	1	0	0	0	1	1
10	10	10	2	0	0	0	1	2

- Fisso $h = [4,4,4,4]^T$ e $w = [4,4,4,4]^T$ e otteniamo un costo di 137.8255.

- Fisso $h = [4, 4, 4, 4]^T$ e $w = [10, 10, 10, 10]^T$ e otteniamo un costo di 333.7739.
- Fisso $h = [2, 2, 2, 2]^T$ e $w = [1, 2, 6, 8]^T$ otteniamo un costo di 144.1224. In questo caso le azioni più privilegiate e utilizzate saranno quelle che implicano una penalità maggiore come si può vedere nell'immagine sottostante.

Action table per penalità crescenti

1	2	3	4	5	6	7	8	9
8	8	8	8	2	3	2	2	1
8	8	8	8	2	3	2	2	2
8	8	8	3	2	3	2	2	3
4	4	4	4	2	3	2	2	4
8	8	8	8	2	3	2	2	5
8	8	8	8	2	3	2	2	6
8	8	8	3	2	3	2	2	7
4	4	4	4	2	3	2	2	8
8	8	8	8	2	3	2	2	9
8	8	8	8	2	3	2	2	10
8	8	8	3	2	3	2	2	11
4	4	4	4	2	3	2	2	12
7	7	7	7	2	3	2	3	1
8	7	7	7	2	3	2	3	2

- Fisso $h = [1, 5, 5, 5]^T$ e $w = [5, 1, 1, 1]^T$. In questo caso abbiamo fatto in modo di avere sul pezzo 1 la massima penalità e il minimo costo di giacenza nel magazzino. Otteniamo che la produzione è concentrata sul pezzo 1 come si può notare in figura. Il costo è pari a 47.8389.

Produzione concentrata su pezzo 1

1	2	3	4	5	6	7	8	9
1	1	1	1	0	0	0	0	1
5	5	5	5	0	0	0	0	2
5	5	5	5	0	0	0	0	3
5	5	5	5	0	0	0	0	4
1	1	1	1	0	0	0	0	5
5	5	5	5	0	0	0	0	6
5	5	5	5	0	0	0	0	7
5	5	5	5	0	0	0	0	8
1	1	1	1	0	0	0	0	9
5	5	5	5	0	0	0	0	10
5	5	5	5	0	0	0	0	11
5	5	5	5	0	0	0	0	12
1	1	1	1	0	0	0	1	1
5	5	5	5	0	0	0	1	2

6) Impatto capacità produttiva

Analizziamo i costi variando la capacità produttiva della linea di produzione, partendo sempre dallo stato iniziale $[0, 0, 0, 0, 1]^T$.
Fissati i seguenti parametri:

$$Imax = [10, 10, 10, 10]^T$$

$$demandProbs = \frac{[1, 1, 1, 1]}{4}$$

$$u = \frac{[1, 1, 1, 1]^T}{2}$$

$$h = [4, 4, 4, 4]^T$$

$$w = [4, 4, 4, 4]^T$$

$$\begin{cases} k = 10F = \sum_{i=1}^{4} h_i \cdot \frac{1}{4} \cdot \frac{1}{k} \cdot \begin{bmatrix} 0 & 0.5 & 1 & 1 \\ 0.5 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0.5 \\ 1 & 1 & 0.5 & 0 \end{bmatrix} \\ F = F - diag\big(diag(F)\big) \end{cases}$$

$$T = 4$$

$$Tb = 1$$

Abbiamo sperimentato diversi casi:

- Capacità produttiva bassa $p = [1, 1, 1, 1]^T$: otteniamo un costo pari a 86.2656.
- Capacità produttiva intermedia $p = [6, 6, 6, 6]^T$, otteniamo un costo pari a 82.5656.

Il primo caso è soggetto a maggiori penalità, ma meno ai costi di giacenza, mentre nel secondo caso le penalità possono essere facilmente ridotte a scapito di un maggiore costo di giacenza. Inoltre, il vantaggio di avere un numero di pezzi prodotti ridotto comporta poter produrre più spesso, in modo da poter rifornire il magazzino quando serve, aspettando un numero minore di istanti che il vincolo di capienza sia rispettato; d'altro canto, produrre di meno in ogni time bucket vuol dire rischiare maggiormente di non soddisfare la domanda.

- Capacità produttiva fuori magazzino $p = [21, 21, 21, 21]^T$, ho un costo totale pari a 90.2000. Esso è dovuto unicamente alla penalità totale sulla domanda, perchè in questo caso nessuna azione di produzione potrebbe essere effettuata poichè nessuna rispetta il vincolo di capacità di magazzino ($p_i > Imax_i, \forall i$).

- Variamo il setting, azzerando il costo di giacenza ed il costo di setup ($h_i, F_{i,j} = 0$), avendo solo le penalità $w = [5, 5, 5, 5]^T$ e le probabilità $demandProbs = \frac{[1,1,1,1,1,1]}{6}$. Simuliamo con produttività crescente $p = [2, 4, 6, 8]^T$, avremo un costo totale uguale a 138.0527. Se vogliamo dare uno sguardo più in dettaglio sulle azioni, possiamo notare che se per esempio prendiamo lo stato con magazzino con capienza rimanente pari alla metà della massima disponibile, possiamo notare che le azioni scelte maggiormente dall'algoritmo sono quelle di setup verso i prodotti 3 e 4, poichè non incidono sul costo (il costo di setup è nullo) e sono quelle che meglio riescono a bilanciare il rispetto dei vincoli di capacità del magazzino e il bisogno di soddisfare la domanda.

Action table per produttività crescenti e penalità fissa

1	2	3	4	5	6	7	8	9
8	8	7	1	5	5	5	5	1
8	2	8	2	5	5	5	5	2
8	8	8	5	5	5	5	5	3
7	7	7	5	5	5	5	5	4
8	8	7	1	5	5	5	5	5
8	2	8	2	5	5	5	5	6
8	8	8	5	5	5	5	5	7
7	7	7	5	5	5	5	5	8
8	8	7	1	5	5	5	5	9
8	2	8	2	5	5	5	5	10
8	8	8	5	5	5	5	5	11
7	7	7	5	5	5	5	5	12
7	7	7	1	5	5	5	6	1
2	2	2	2	5	5	5	6	2

- Stesso setting di parametri fissi dell'ultimo esperimento, scegliendo $demandProbs = \frac{[1,1,1,1,1,1,1,1,1,1,1]}{11}$, $w = [2, 4, 6, 8]^T$, e $p = [2, 4, 6, 8]^T$, otteniamo un costo di 286.7066. Per quanto riguarda le azioni, se prendiamo ad esempio lo stato con magazzino vuoto, si ottiene che avendo molta più penalità sui prodotti ad alta capacità produttiva, si avrà sopratutto un incentivo molto forte a produrre questi ultimi.

Action table per produttività e penalità crescenti

1	2	3	4	5	6	7	8	9
8	8	8	8	5	5	5	5	1
8	8	8	8	5	5	5	5	2
8	8	8	8	5	5	5	5	3
7	7	7	7	5	5	5	5	4
8	8	8	8	5	5	5	5	5
8	8	8	8	5	5	5	5	6
8	8	8	8	5	5	5	5	7
7	7	7	7	5	5	5	5	8
8	8	8	8	5	5	5	5	9
8	8	8	8	5	5	5	5	10
8	8	8	8	5	5	5	5	11
7	7	7	7	5	5	5	5	12
8	8	8	8	5	5	5	6	1
8	8	8	8	5	5	5	6	2