POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria Matematica

Tesi di Laurea Magistrale

Modelli MILP stocastici e mateuristiche per problemi di lot sizing

Relatori
prof. Paolo Brandimarte
firma del relatore

. . . . . .

Candidato
Paolo Gorino
firma del candidato

. . . . . .

Anno Accademico 2022-2023

A Leonardo

Sommario

La tesi tratta la risoluzione del problema di lot sizing caratterizzato da una disponibilità massima in termini di tempo, sia nel caso di modello deterministico che in quello di domanda stocastica. Si è sperimentato un metodo di riduzione della varianza e la mateuristica 'Fix and Relax' per ridurre la lunga durata necessaria a risolvere un modello di ottimizzazione misto intero.

Ringraziamenti

Come in occasione della laurea triennale, il mio ringraziamento principale va ai miei genitori, Anna Maria e Aldo: grazie per avermi sempre sostenuto, supportato e sopportato sempre, nei momenti belli ed in quelli meno belli, indipendentemente dal fatto che foste d'accordo o meno con le mie azioni e decisioni.
Grazie ad Andrea e Claudia per i consigli, per l'interessamento costante, per avermi scelto come loro testimone e soprattutto per Leonardo, a cui è dedicata questa tesi.
Grazie anche a tutto il resto della mia famiglia, zii/e, prozii/e vari e cugini di ogni grado.
Grazie a Luca e Simone, amici da una vita, per tutte le serate insieme, per tutte le birre bevute a Mombello, per tutto quello che abbiamo fatto insieme in passato e per quello che faremo in futuro.
Grazie a tutti i miei amici, principalmente i più presenti in questi ultimi anni, Pier, Marco, Mario, Matteo, Luca, Gabriele, Luca, Davide, Monica, Erika, Francesco, Alessandro, Michele, Fabio, Alessio, Fabrizio, Davide, Sara, Denise, per tutti i bei momenti che ho avuto la fortuna di passare con loro.
Grazie anche a tutti i miei ex compagni di classe delle superiori, perché raramente mi sono trovato bene come gli ultimi anni in quella classe, a tutti i miei compagni, allenatori, dirigenti (passati e presenti) di calcio alla Buttiglierese, per i traguardi raggiunti e non, in particolare per quella promozione.
Grazie a tutti i ragazzi, ed ora le ragazze, che ho allenato in questi anni a Buttigliera, per la spensieratezza e le soddisfazioni che mi hanno regalato.
Grazie infine al professore Paolo Brandimarte per il modo efficace e coinvolgente di attirare l'attenzione nell'insegnare (a cui mi sono ispirato per interagire con i ragazzi/e che alleno), che mi ha spinto a contattarlo per la tesi, e ad Edoardo Fadda e Daniele Giovanni Gioia per l'aiuto che mi hanno dato con gli argomenti nuovi che ho affrontato nello svolgimento di questo elaborato.
Grazie infine a chiunque mi abbia aiutato in qualsiasi modo in questo periodo, un grazie ancora più grande a chi mi fossi dimenticato di menzionare, sappia che il motivo è stato la fretta di concludere l'elaborato, non certo una sua minore importanza.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione generale

1.1 Il problema CLSP

L'argomento principale della tesi è la risoluzione del problema di lot sizing caratterizzato da una capacità massima (dall'inglese Capacitated Lot-Sizing Problem, talvolta nel seguito abbreviato con CLSP), sia in termini di tempo che in termini di quantità producibile. Il tempo viene discretizzato in una serie di istanti temporali in cui si prende una decisione di produzione, si osserva la domanda e si aggiornano di conseguenza le quantità presenti nel magazzino. Dato un numero arbitrario di prodotti, in questa trattazione per ognuno di essi si considera un costo di magazzino, un costo per il non soddisfacimento della domanda ed un costo di setup della macchina che lo produce. Il costo di magazzino unitario rappresenta la spesa di mantenere un singolo pezzo di prodotto, non venduto, a magazzino per un istante temporale. Il costo, o meglio la penalità, per la domanda non soddisfatta rappresenta la spesa, nel senso di mancato guadagno, per ogni singolo pezzo che non si è riuscito a vendere in un istante temporale perché la disponibilità del pezzo era inferiore alla domanda. Infine il costo di setup rappresenta la spesa per l'utilizzo della macchina che produce il pezzo del prodotto in un istante temporale. Esso è indipendente dal numero di pezzi prodotti, dipende solo dal fatto che nel determinato istante di tempo si produca o meno. La funzione obiettivo del modello è la somma dei costi sull'orizzonte temporale deciso, e va ovviamente minimizzata.
L'incertezza della domanda sarà rappresentata in un paio di modi differenti, utilizzando un albero di scenari (caso stocastico) oppure andando a mettere delle scorte di sicurezza al livello dei prodotti a magazzino (caso deterministico).
Il modello, che nel seguito verrà riportato in modo più preciso a seconda delle diverse formulazioni utilizzate, prevede anche un vincolo di bilanciamento del flusso di ogni prodotto ad ogni istante di tempo, un vincolo sulla quantità massima di ogni pezzo che può essere prodotta ed un vincolo sulla disponibilità massima in termini di tempo.
Le variabili sono di tipo continuo e a valori maggiori o uguali a zero (quantità prodotta di ogni pezzo in ogni istante, quantità in magazzino di ogni prodotto in ogni istante, eventualmente domanda non soddisfatta di ogni prodotto in ogni istante), ad eccezione delle variabili di setup, che possono assumere solo i valori interi 0 (nel caso in cui l'articolo

non sia stato prodotto nell'istante temporale) oppure 1 (nel caso in cui invece sia stato prodotto). Si tratta dunque di un modello di tipo misto-intero, di cui è noto l'elevato costo computazionale della risoluzione.

1.2 Simulazioni rolling horizon

A partire dai modelli, descritti in modo sommario nella precedente sezione, durante il lavoro sono stati definiti i rispettivi solver che, data una situazione iniziale di magazzino ed un elenco dei parametri del problema (i valori dei costi e delle capacità, eventualmente un albero di scenari o le scorte di sicurezza), calcola e restituisce le decisioni al primo stadio, ovvero le quantità di ogni articolo da produrre all'istante temporale in questione.
Per confrontare i risultati out of sample di questi diversi solver, si è scelto di utilizzare una struttura di simulazioni di tipo rolling horizon. Essa consiste nel campionare alcuni "percorsi" di domanda, ovvero una serie di realizzazioni della variabile aleatoria che si è utilizzata per rappresentarla. Su ognuno di questi percorsi, istante temporale per istante temporale, si utilizza il solver sotto esame per calcolare la quantità da produrre di ogni articolo, e i conseguenti valori delle variabili di setup. In seguito si osserva la domanda dal percorso, si aggiorna il numero di pezzi in magazzino (dato il magazzino dell'istante precedente, quanto si è prodotto e la domanda osservata) ed eventualmente la domanda che non è stata soddisfatta. Da questi valori, assieme alle variabili di setup restituite dal solver, si calcola il costo dell'istante temporale in questione. Le medesime operazioni vanno ripetute fino ad arrivare alla fine del percorso di domanda, andando a sommare in modo cumulativo i costi di ogni istante, fino ad ottenere il costo totale del percorso. Fatto ciò per ogni percorso, si è usata la media dei costi ottenuti come valore per confrontare le prestazioni dei diversi solver sperimentati.

1.3 Mateuristica Fix & Relax

Dalla descrizione della struttura di simulazioni di tipo rolling horizon, risulta chiaro che il modello misto intero, rapidamente introdotto nella prima sezione dell'introduzione, va risolto numerose volte, una per ogni istante temporale dei percorsi. Questi devono essere abbastanza lunghi, in modo che i costi totali siano significativi, e non ad esempio dovuti al valore iniziale del magazzino.
Nel caso in cui, come faremo, l'incertezza della domanda venga rappresentata da un albero di scenari, il modello misto intero diventa di larga scala. Come è noto, questo problema è di tipo NP-hard, il che rende impossibile pensare di risolverlo in maniera ottimale per numerose volte, come richiesto dalle simulazioni rolling horizon.
Per diminuire il costo computazionale, e rendere di conseguenza percorribile la struttura di simulazione scelta, ci si deve accontentare di una soluzione sub-ottimale, usando delle strategie di soluzione euristiche. In particolare, in questo elaborato si mette in pratica la mateuristica Fix & Relax. Essa verrà descritta nel dettaglio in seguito, per ora ci si limita a esporre l'idea alla sua base. Il problema misto intero viene risolto più volte iterativamente, rilassando a valori continui le variabili intere, tranne un loro sottoinsieme che resta a valori in $\{0,1\}$. Ad ogni iterazione, le variabili intere calcolate in quella

precedente vengono fissate al valore della soluzione, dunque si riduce il numero di variabili. Questo rilassamento delle variabili rende il tempo di soluzione molto inferiore rispetto al caso del modello effettivo, ma ovviamente la soluzione che si ottiene non è ottimale. Nel seguito, dopo avere descritto nel dettaglio il modo in cui si sono realizzati i rispettivi solver, si andranno a confrontare le prestazioni per vedere e quantificare i pro ed i contro dell'utilizzo della mateuristica.

Capitolo 2

Corpo del rapporto

2.1 Il problema CLSP stocastico ed il solver standard

Per andare a formalizzare in maniera precisa il modello, bisogna prima fare alcune ipotesi. Innanzitutto si suppone che la domanda dei prodotti ad ogni istante di tempo (o in ogni nodo, visto che a breve si parlerà di alberi di scenari) sia indipendente ed identicamente distribuita come una normale multivariata. Nel caso in cui alcune componenti estratte assumessero valori negativi (cosa possibile, visto la distribuzione scelta, ma priva di senso, visto che si vuole rappresentare una domanda), a quelle componenti si associa il valore 0. Per rappresentare l'incertezza della domanda, come anticipato, si è scelto di utilizzare un albero di scenari, in linea con quanto fatto nell'articolo Brandimarte [2006b]. Si riporta un esempio grafico di albero di scenari (figura 2.1), per poterne descrivere alcune caratteristiche rilevanti nel seguito.

L'albero è una discretizzazione di una distribuzione di probabilità più complessa, ad ogni suo nodo è associata una realizzazione della domanda. Il nodo 0 si dice nodo radice, è l'unico a non avere predecessori e rappresenta lo stato attuale, quello in cui bisogna prendere la decisione delle quantità da produrre.
L'albero in questo caso è composto da 4 periodi, il nodo 0 è l'unico dell'istante temporale $t = 0$, i nodi 1, 2 e 3 sono all'istante $t = 1$ e così via. L'albero è caratterizzato dai propri fattori di ramificazione (in inglese branching factors), che in questo caso sono [3,2,1]. Questo significa che il nodo radice ha tre successori, ognuno dei quali ha a sua volta due successori, ognuno dei quali ha a sua volta un successore. Nel caso di fattore di ramificazione unitario, si usa solitamente il valore atteso della distribuzione come valore della domanda nel singolo nodo. Una sequenza di nodi (da nodo radice ad una foglia) definisce uno scenario, per esempio in questo albero sono presenti 6 scenari.
Ad ogni ramificazione dell'albero di scenari è associata una probabilità condizionata, che non deve necessariamente essere uniforme, ma che nel seguito considereremo sempre tale per semplicità. É dunque immediato calcolare le probabilità di ogni nodo, le quali verranno utilizzate per definire un valore atteso dei costi totali, che costituirà la funzione obiettivo del modello, ovviamente da minimizzare.
Per la scrittura del modello serve definire le seguenti notazioni, denotate in maniera fedele

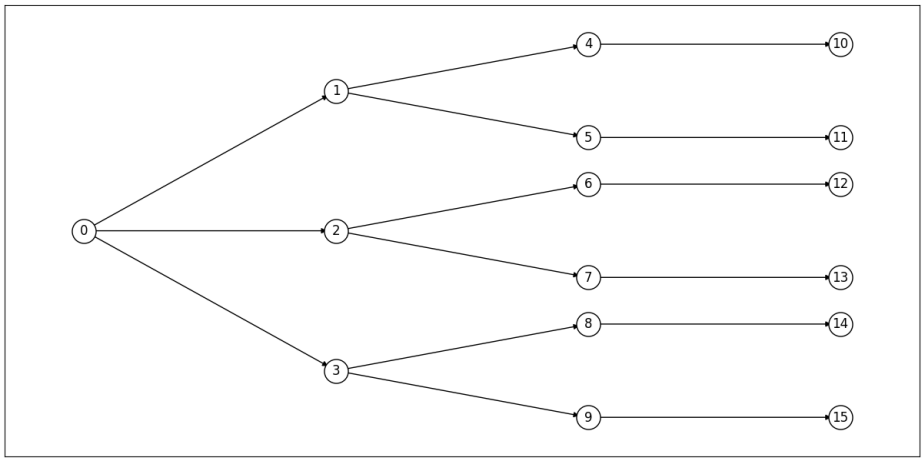

Figura 2.1. Esempio albero di scenari

all'articolo:

- $n \in N$ denota un generico nodo dell'albero; T rappresenta l'insieme dei nodi terminali, o foglie, dell'albero; nell'esempio $N = \{0,1,...,15\}$ e $T = \{10,11,...,15\}$
- T(n) rappresenta l'istante di tempo del nodo n, nell'albero della figura T(0)=0 e T(12)=3
- $a(n)$ è il predecessore del nodo n, nell'esempio $a(6) = 2$
- $\Omega(n,t)$ è l'unico "antenato" del nodo n all'istante di tempo t, con $t <$T(n); nell'esempio $\Omega(15,1) = 3$
- $\Sigma(n,t)$ è l'insieme dei successori del nodo n all'istante di tempo t, con $t >$T(n); nell'esempio $\Sigma(2,3) = \{12,13\}$

Un altro aspetto da delineare è il profilo dell'incertezza della domanda: come nell'articolo precedentemente citato si suppone di scoprire la domanda dell'istante temporale prima di prendere la decisione di produzione e di avere incertezza costante e totale per gli istanti successivi. Questa ipotesi è un'estremizzazione (ma che semplifica la formulazione) del fatto, ragionevole, che negli istanti iniziali di tempo gli ordini dei clienti tendono ad essere abbastanza completi, dunque vi è poca incertezza sulla domanda, mentre in quelli distanti nel tempo è necessario affidarsi di più a previsioni, quindi con incertezza maggiore.
Per andare a descrivere il modello, si utilizza una formulazione di tipo plant location. Essa consiste nel disaggregare la variabile $x_{i,t}$ (ovvero la quantità dell'articolo i prodotta all'istante di tempo t) nelle variabili $y_{i,t,p}$, che rappresentano la quantità del prodotto i,

prodotta al tempo t per soddisfare la domanda nell'istante di tempo futuro p, con $p \geq t$. Questa scelta, seppur complichi la definizione del modello nel caso di domanda stocastica rappresentata da un albero di scenari, riduce il gap di integralità rispetto alle formulazioni naturali del modello, ovvero quelle con le variabili non disaggregate. Questo comporta una riduzione dei nodi da esplorare dall'algoritmo di risoluzione branch and bound e dunque una maggiore velocità.
Fatte queste premesse, veniamo a definire nello specifico il modello CLSP stocastico:

$$min \quad \sum_{n \in N} p^{[n]} \Big[\sum_i \Big(f_i s_i^{[n]} + h_i I_i^{[n]} + g_i z_i^{[n]} \Big) \Big] + \sum_{n \in N \setminus T} p^{[n]} \Big[\sum_i \sum_{\tau > T(n)} h_i \Big(\tau - T(n) \Big) y_{i,\tau}^{[n]} \Big] \tag{2.1}$$

$$s.t. \quad I_i^{[a(n)]} + \sum_{t < T(n)} y_{i,T(n)}^{[\Omega(n,t)]} + y_{i,T(n)}^{[n]} = d_i^{[n]} + I_i^{[n]} - z_i^{[n]} \qquad \forall i, n \tag{2.2}$$

$$y_{i,\tau}^{[n]} \leq \Big(\max_{j \in \Sigma(n,\tau)} d_i^{[j]} \Big) s_i^{[n]} \qquad \forall i, n, \tau > T(n) \tag{2.3}$$

$$y_{i,T(n)}^{[n]} \leq d_i^{[n]} s_i^{[n]} \qquad \forall i, n \tag{2.4}$$

$$\sum_i \sum_{\tau \geq T(n)} r_i y_{i,\tau}^{[n]} + \sum_i r_i' s_i^{[n]} \leq R \qquad \forall n \tag{2.5}$$

$$y_{i,\tau}^{[n]}, I_i^{[n]}, z_i^{[n]} \geq 0 \tag{2.6}$$

$$s_i^{[n]} \in \{0,1\} \tag{2.7}$$

Nel seguente elenco si riportano i significati delle variabili e dei parametri del modello:

- $y_{i,\tau}^{[n]}$ è la quantità del prodotto i, prodotta nel nodo n per soddisfare la domanda del periodo di tempo futuro τ, con $\tau \geq T(n)$
- $I_i^{[n]}$ è la quantità di prodotto i che resta in magazzino nel nodo n (eventualmente 0) e che viene passata ai suoi immediati successori nell'albero
- $z_i^{[n]}$ è la quantità di domanda del prodotto i non soddisfatta al nodo n (eventualmente 0)
- $s_i^{[n]}$ è la variabile di setup per il prodotto i nel nodo n
- $p^{[n]}$ è la probabilità di occorrenza del nodo n
- f_i è il costo di setup per il prodotto i
- h_i è il costo di magazzino per il prodotto i
- g_i è la penalità per la domanda non soddisfatta per il prodotto i
- $d_i^{[n]}$ è la domanda del prodotto i nel nodo n
- r_i è il tempo di lavorazione del singolo pezzo del prodotto i

- $r_i^{'}$ è il tempo di setup per il prodotto i
- R è la capacità massima in termini di tempo

La funzione obiettivo 2.1 del modello è il valore atteso dei costi ed è divisa in due termini: il primo rappresenta il valore atteso dei costi di setup, di magazzino in eccesso (ciò che si è prodotto in eccesso rispetto alla domanda e resta in magazzino per un periodo) e delle vendite perse. Il secondo termine è il valore atteso del costo del magazzino (ovvero i costi di tenere a magazzino ciò che si produce per istanti di tempo successivi al nodo in questione), ed è presente solo per i nodi non terminali (nei nodi terminali non si produce intenzionalmente per il futuro). Non è presente alcun termine di valore del magazzino finale.
Il vincolo 2.2 è un bilanciamento del flusso per ogni nodo dell'albero di scenari ed ogni prodotto. Il flusso entrante è dato dalla somma del magazzino in eccesso del nodo predecessore, di tutte le produzioni in periodi precedenti destinate al periodo del nodo in questione e di ciò che viene prodotto nel nodo per il consumo immediato. Il flusso uscente invece è dato dalla somma della domanda nel nodo, del magazzino rimasto in eccesso meno le vendite perse. Questo vincolo andrà poi scritto separatamente per il nodo radice dell'albero, in quanto esso non ha predecessori/antenati.
Il vincolo 2.3 indica una capacità massima di produzione per ogni nodo ed ogni oggetto. Come capacità massima si deve scegliere il massimo delle domande dei possibili nodi successori al periodo di tempo per cui stiamo producendo.
Nel caso invece di produzione per il consumo immediato, per l'ipotesi fatta in precedenza, la domanda è nota e si usa quella come capacità massima di produzione, come si vede nel vincolo 2.4.
Il vincolo 2.5 impone che, in ogni nodo, la somma dei tempi di lavorazione degli articoli prodotti e dei tempi di setup delle relative macchine non superino la capacità massima in termini di tempo R.
Infine i vincoli 2.6 e 2.7 impongono alle variabili continue di non assumere valori negativi e alle variabili di setup di essere a valori interi in {0,1}.
A partire da questo modello, si è definita la classe Python CLSP_S. Essa serve per definire e risolvere il problema del lot sizing soggetto a capacità, con domanda di tipo stocastico rappresentata da un albero di scenari. Oltre al classico metodo ___init___, questa classe contiene tre metodi, il più importante dei quali è il metodo solve, di cui si riporta in seguito il codice:

```
def solve(
    self, instance, tree, I0, time_limit=None, gap=None,
    verbose=False):

    model = self.populate(instance, tree, I0)
    return self.get_solution(instance, model,
    time_limit=time_limit, gap=gap, verbose=verbose)
```

Questo metodo richiama al suo interno gli altri due della classe: il metodo populate, dato un elenco dei parametri (costi, capacità) del problema (contenuto in instance), un albero di scenari ed un valore del magazzino iniziale, definisce il modello descritto formalmente

in precedenza. Questo modello, insieme all'elenco dei parametri, e ad altri parametri opzionali (time_limit, gap e verbose) sono passati in ingresso al metodo get_solution, che risolve il modello e restituisce in uscita il valore della funzione obiettivo, le decisioni al primo stadio (quantità da produrre di ogni articolo e variabili di setup), il tempo necessario alla soluzione del modello e la distanza della soluzione trovata rispetto a quella ottima.

2.2 Mateuristica Fix & Relax e relativo solver

Come anticipato nell'introduzione, il modello descritto nella sezione precedente è di tipo misto intero e, per problemi realistici, di larga scala. Il solver definito in precedenza, che utilizza il classico algoritmo branch and bound per risolvere il problema, impiega tempi decisamente troppo lunghi per trovare una soluzione, dunque, dovendo risolvere questo tipo di modello molte volte all'interno delle simulazioni rolling horizon, è necessario trovare un modo per renderli più brevi. Occorre quindi accettare una soluzione sub-ottimale a cui si arriva utilizzando approcci di soluzione di tipo euristico. In questa tesi in particolare si sperimenta la mateuristica Fix & Relax.
L'idea alla sua base è quella di partizionare le variabili intere in sottoinsiemi più piccoli e risolvere una sequenza di sotto problemi misto interi, in ognuno dei quali uno dei sottoinsiemi di variabili resta a valori interi e gli altri sono rilassati a valori continui. Questi sotto problemi sono a loro volta misto interi, ma con un minor numero di variabili intere, su di essi l'algoritmo di risoluzione branch and bound risulta quindi molto più veloce. Nel primo sotto problema, solo il primo sottoinsieme di variabili intere rimane a valori interi, tutti gli altri sono rilassati a valori continui. Il modello viene risolto e le variabili intere vengono fissate al valore ottenuto nella soluzione. Nel secondo sotto problema dunque il primo sottoinsieme di variabili intere è fissato ai valori calcolati (non sono dunque più variabili), il secondo sottoinsieme viene lasciato a valori interi, mentre tutti gli altri sono rilassati a valori continui. Si risolve questo secondo modello e il secondo gruppo di variabili intere viene fissato ai valori calcolati. Queste operazioni si ripetono fino all'ultimo sottoinsieme di variabili intere ed alla risoluzione del relativo sotto problema.
Bisogna dunque decidere come partizionare le variabili intere, che nel caso del problema del lot sizing sono le variabili di setup, una per ogni combinazione di nodo e prodotto. Il modo più naturale per partizionarle è rispetto all'istante di tempo del nodo a cui si riferiscono.
Supponendo che l'albero di scenari sia composto da T istanti di tempo, dall'istante 0 (quello del solo nodo radice) all'istante T-1 (quello delle foglie), l'algoritmo di risoluzione Fix & Relax segue il seguente schema:

1. fissare k=0;

2. risolvere con l'algoritmo branch and bound il sotto problema in cui le variabili di setup relative a nodi n tali che $T(n) < k$ sono fissate e quelle relative a nodi n tali che $T(n) > k$ sono rilassate a valori continui

3. aggiornare k=k+1 e ripetere il passo 2 fino a che $k < T$

In particolare, per quanto riguarda il passo 2, sono possibili tre diverse situazioni che ora verranno esposte per chiarezza:

- $k = 0$
  - Assenti variabili già fissate
  - $s_i^{[0]} \in \{0,1\} \quad \forall i$
  - $s_i^{[n]} \in [0,1] \quad \forall i, \forall n : T(n) > 0$
- $0 < k < T - 1$
  - $s_i^{[n]}$ fissate $\quad \forall i, \forall n : T(n) < k$
  - $s_i^{[n]} \in \{0,1\} \quad \forall i, \forall n : T(n) = k$
  - $s_i^{[n]} \in [0,1] \quad \forall i, \forall n : T(n) > k$
- $k = T - 1$
  - $s_i^{[n]}$ fissate $\quad \forall i, \forall n : T(n) < T - 1$
  - $s_i^{[n]} \in \{0,1\} \quad \forall i, \forall n : T(n) = T - 1$
  - Assenti variabili rilassate a valori continui

A partire dal modello esposto nella sezione precedente e da queste differenti definizioni delle variabili di setup, si è definita la classe Python CLSP_FR, che serve a risolvere un problema CLSP mettendo in pratica la mateuristica Fix & Relax. Essa è molto simile alla classe precedente, riportiamo in seguito le differenze, a partire dal metodo solve:

```
def solve(
    self, instance, tree, I0, time_limit=None, gap=None,
    verbose=False):

    #NUMERO DI ISTANTI DI TEMPO DELL'ALBERO DI SCENARI
    time_periods=instance.time_periods
    #NUMERO DI PRODOTTI
    num_items=instance.num_items
    #NUMERO DI NODI
    N=instance.N
    #MATRICE IN CUI SI SALVANO LE VARIABILI INTERE GIA' FISSATE
    fixed=np.zeros((num_items,N))
    T=time_period(tree)
    tot_time=0

    #AD OGNI ISTANTE DI TEMPO "SI POPOLA" IL MODELLO,
    #PER AGGIORNARE LE VARIABILI FISSE, QUELLE BINARIE E
    #QUELLE RILASSATE
    for t in range(time_periods):
```

```
            model = self.populate(instance, tree, I0, t, fixed)
            [of,Y,S,comp_time,computed_s]=self.get_solution(instance,
                model, tree, t, time_limit=time_limit, gap=gap,
                verbose=verbose)

            i1=[i for i in range(len(T)) if T[i]==t]
            n1=min(i1)
            #LE VARIABILI CHE ERANO BINARIE A QUESTA ITERAZIONE
            #VENGONO "FISSATE" AL VALORE CHE E' STATO CALCOLATO
            #(NELLA MATRICE fixed). ALL'ITERAZIONE SUCCESSIVA
            #SARANNO FISSE
            for k in i1:
                fixed[:,k]=computed_s[:,k-n1]
            #SOMMA DEL TEMPO COMPUTAZIONALE DEL SOTTO PROBLEMA
            #DELL'ISTANTE t A QUELLI PRECEDENTI
            #(PER OTTENERE TEMPO TOTALE)
            tot_time=tot_time+comp_time

    return of,Y,S,tot_time
```

Analogamente al caso precedente in solve sono richiamati gli altri due metodi della classe, populate (per definire il modello) e get_solution (per risolverlo). La differenza è che vengono chiamati all'interno di un ciclo for, dunque molteplici volte, una per ogni istante di tempo dell'albero di scenari passato in ingresso. Questo succede perché la mateuristica Fix & Relax prevede la risoluzione dei sotto problemi con un sottoinsieme di variabili intere e le altre fisse o rilassate a valori continui. Ogni volta che un sotto problema viene risolto, il sottoinsieme di variabili intere viene fissato ai valori che sono stati calcolati, i quali vengono salvati all'interno della matrice fixed. Questa matrice, inizialmente di soli 0 in quanto non sono ancora state fissate variabili, è ad ogni iterazione un ingresso del metodo populate, assieme all'istante temporale t che caratterizza il sotto problema.
Il metodo populate, a differenza del caso precedente, dipende da questi due parametri che vengono passati in ingresso: a seconda dell'istante temporale t, sono definite in maniera differente le variabili intere e quelle rilassate. Si riporta in seguito un frammento del codice (quello che copre il caso intermedio descritto prima, ovvero $t = k$ con k compreso tra 0 e $T - 1$).

```
#insieme degli indici dei nodi il cui periodo temporale e'
#uguale a t
i1=[i for i in range(len(T)) if T[i]==t]
#insieme degli indici dei nodi il cui periodo temporale e'
#maggiore di t
i2=[i for i in range(len(T)) if T[i]>t]

#numero di nodi il cui periodo temporale e' uguale a t
#(numero di variabili di set up intere)
```

```
Nt=len(i1)
#numero di nodi il cui periodo temporale e' maggiore di t
#(numero di variabili di set up rilassate)
Ngt=len(i2)

#indice del primo nodo il cui periodo temporale e'
#uguale a t
n1=min(i1)
#indice del primo nodo il cui periodo temporale e'
maggiore di t
n2=max(i1)+1

...

else:

    #VARIABILI DI SET UP INTERE
    s = model.addMVar(
        shape=(num_items,Nt),
        vtype=grb.GRB.BINARY,
        name="s"
    )

    self.s=s

    #VARIABILI DI SET UP RILASSATE
    relaxed = model.addMVar(
        shape=(num_items,Ngt),
        vtype=grb.GRB.CONTINUOUS,
        lb=0,
        ub=1,
        name="relaxed"
    )

    self.relaxed=relaxed
```

Le variabili di setup vengono definite separatamente: quelle denotate con s sono quelle facenti parte del sottoinsieme di variabili intere (e sono in numero $num\_items \times Nt$, dove Nt, che viene calcolato ogni volta che si chiama il metodo populate con un diverso t, rappresenta il numero di nodi il cui istante temporale è pari a t), mentre quelle denotate con relaxed sono quelle rilassate a valori interi (analogamente, sono in numero $num\_items \times Ngt$, dove Ngt è il numero di nodi il cui istante temporale è maggiore di t).
Infine, l'altra differenza rispetto al metodo populate della classe precedente è la scrittura dei vincoli che riguardano le variabili di setup: poiché esse sono rappresentate da tre tipi

di variabili diverse (sempre nel caso di t intermedio, quelle fissate dalla matrice fixed, quelle intere dalla variabile s e quelle rilassate dalla variabile relaxed), i vincoli vanno scritti separatamente per i tre tipi. Si riporta di seguito il frammento di codice che fissa il vincolo 2.4 per il caso di t intermedio:

```
model.addConstrs((y[i,T[n],n]<=d[i,n]*fixed[i,n])
    for i in range(num_items) for n in range(n1))

model.addConstrs((y[i,T[n],n]<=d[i,n]*s[i,n-n1])
    for i in range(num_items) for n in range(n1,n2))

model.addConstrs((y[i,T[n],n]<=d[i,n]*relaxed[i,n-n2])
    for i in range(num_items) for n in range(n2,N))
```

Il singolo vincolo viene formulato in tre diverse righe di codice, una per ogni tipo di variabile. Da notare anche il fatto che le variabili y e le domande d sono indicizzate, per quanto riguarda l'indice dei nodi, da 0 a $N-1$ (se N è il numero di nodi dell'albero). Le variabili di setup (quelle intere e quelle rilassate a valori continui) sono ad ogni iterazione indicizzate da 0 a $Nt-1$ e da 0 a $Ngt-1$ rispettivamente. É dunque necessario 'scalare' nel modo corretto l'indice della variabile di setup all'interno dei vincoli, sottraendo $n1$ ed $n2$ (rispettivamente per le variabili di setup intere e per quelle rilassate) all'indice dei nodi n. Come si vede dal listato di codice precedente, $n1$ ed $n2$ sono rispettivamente l'indice del primo nodo il cui istante temporale è uguale a t e l'indice del primo nodo il cui istante temporale è maggiore di t.

2.3 Risultati in sample dei due solver

Dopo aver definito i due diversi solver, quello standard e quello che utilizza la mateuristica Fix & Relax, è naturale interrogarsi su come si comportino sullo stesso problema, su quanto sia sub ottimale la soluzione trovata e al contempo su quanto tempo si risparmi usando la mateuristica. Per farlo si è dunque risolto un numero di istanze del problema (definite usando diversi fattori per alcuni parametri), utilizzando su ognuno di esse i due solver, e si sono confrontati i risultati.
Innanzitutto, nelle due sezioni precedenti si è visto che il metodo solve di entrambi i solver richiede in ingresso un albero di scenari, per rappresentare l'incertezza della domanda. A partire dalla classe DiGraph di NetworkX (Aric A. Hagberg and Swart [2008]), si è quindi definita una classe ScenarioTree che crea un albero di scenari, le cui osservazioni ad ogni nodo sono campionate da una normale multivariata, che è la distribuzione che si è scelta per descrivere la domanda. Si riportano in seguito un paio di frammenti di codice che mostrano il funzionamento della classe:

```
self.add_node(
    self.starting_node,
    obs=initial_value,
    prob=1,
```

```
        t=0,
        id=0,
        stage=0
    )
    self.name = name
    self.breadth_first_search = []
    self.depth = len(branching_factors)
    self.branching_factors = branching_factors
    # Calcolo del numero di scenari totale
    self.n_scenarios = prod(self.branching_factors)
    count = 1
    last_added_nodes = [self.starting_node]
    n_nodes_per_level = 1
```

In questo primo frammento si crea il nodo radice, in cui si osserva il valore passato in ingresso initial_value (che per le ipotesi fatte è il valore atteso della distribuzione normale multivariata), e si salvano alcune variabili utili per inserire i successivi nodi, tra cui la profondità dell'albero, il suo fattore di ramificazione, il numero totale di scenari, il numero corrente di nodi, la lista dei nodi aggiunti all'ultimo istante di tempo.

```
for i in range(self.depth):
    next_level = []
    n_nodes_per_level *= self.branching_factors[i]
    # per ogni nodo 'genitore' si aggiungono nodi 'figli'
    for parent_node in last_added_nodes:
        # probabilita' uniforme ad ogni ramificazione
        probs=1/self.branching_factors[i]
        for j in range(self.branching_factors[i]):
            id_new_node = count

            # se fattore di ramificazione e' 1, nel nodo
            # si osserva il valore atteso
            if(self.branching_factors[i]==1):
                sample=mean[0,:]
            # altrimenti, campionamento da normale multivariata
            else:
                sample=np.random.multivariate_normal(mean[0,:], cov)
                for k in range(len(sample)):
                    if sample[k]<0:
                        sample[k]=0

            self.add_node(
                id_new_node,
                obs=sample,
                prob=self.nodes[parent_node]['prob'] * probs,
                t=i + 1,
```

```
                id=count,
                stage=i + 1
            )
            self.add_edge(parent_node, id_new_node)
            next_level.append(id_new_node)
            count += 1
    last_added_nodes = next_level
    self.n_nodes = count
self.leaves = last_added_nodes
```

Per aggiungere gli altri nodi si usano tre cicli for. Il primo scorre i valori da 0 alla profondità dell'albero, per popolare tutti i livelli necessari. Il secondo scorre i nodi presenti nel livello precedente, per assegnare i nodi 'figli' a tutti i nodi 'genitori'. Il terzo ciclo scorre i valori da 0 al valore del vettore dei fattori di ramificazione corrispondente alla profondità a cui ci si trova, per creare il giusto numero di nodi 'figli' per ogni nodo 'genitore'.
All'interno dei tre cicli for si crea il nuovo nodo con il metodo .add_node. Si campiona il valore osservato da una normale multivariata con la funzione random.multivariate_normal, a meno che il nodo sia figlio di un fattore di ramificazione pari ad 1, nel qual caso si osserva il valore atteso della distribuzione. Nel caso in cui qualche componente estratta fosse negativa, la si sostituisce con il valore 0 per ipotesi. Si aggiunge quindi il nuovo nodo, assegnandogli il valore osservato, la sua probabilità (per le ipotesi fatte uniforme, si noti dal listato la definizione di *probs*), l'istante di tempo e il suo id (numero intero che identifica il nodo, il nodo radice è il nodo 0, gli altri vengono identificati con valori crescenti in ordine di inserimento nell'albero). Si crea quindi un arco tra il nodo appena creato ed il suo nodo 'genitore', si aggiunge il nuovo nodo alla lista di quelli definiti nel livello in corso e si aumenta di 1 il conto dei nodi dell'albero.
All'iterazione (sulla profondità dell'albero) successiva i nodi aggiunti nell'ultimo livello diventano nodi 'genitori'. Quando anche l'ultima iterazione del ciclo più esterno è conclusa, i nodi aggiunti all'ultimo livello sono le foglie dell'albero.
Uno degli ingressi dell'albero di scenari è il fattore di ramificazione: si è scelto di esaminare i 5 diversi branching factors [5,3,2], [6,5,2], [5,3,2,2], [9,5,2,1], [5,3,3,2,1].
Per ognuno di essi sono stati definiti e risolti molteplici problemi (con il solver standard e con quello Fix & Relax), facendo variare i seguenti fattori:

- costi di magazzino (h_fact):
  - h_fact=0: costi di magazzino bassi, numero intero estratto uniformemente tra 1 e 4
  - h_fact=1: costi di magazzino alti, numero intero estratto uniformemente tra 7 e 10
- penalità per domanda non soddisfatta (g_fact):
  - g_fact=0: penalità per domanda non soddisfatta bassa, numero intero estratto uniformemente tra 50 e 70
  - g_fact=1: penalità per domanda non soddisfatta alta, numero intero estratto uniformemente tra 80 e 100

- tempo medio tra ordini (TBO):
  - TBO=2
  - TBO=3
  - TBO=4
- 10 diversi seed per l'albero di scenari

Il T_{bo} viene utilizzato per generare i costi di setup, secondo la formula

$$f_i = \frac{D_i}{2}(T_{bo})^2 h_i,$$

in cui D_i rappresenta il valore atteso della domanda per il prodotto i, ed h_i il suo costo di magazzino.
Gli altri parametri del modello, che sono stati mantenuti fissi al variare delle combinazioni dei fattori appena illustrati, sono stati generati nel modo seguente:

- il valore atteso della domanda per ogni prodotto è un numero intero estratto in maniera uniforme tra 10 e 100. Il coefficiente di variazione per ogni prodotto è estratto in maniera uniforme tra 0.1 e 0.8. A partire da questi valori si calcolano le varianze delle domande. Per generare la matrice di varianza-covarianza, si estrae una matrice di correlazione (con elementi pari ad 1 sulla diagonale e con elementi extra diagonali estratti in maniera uniforme tra -1 ed 1, in modo che la matrice sia simmetrica). Date le varianze e le correlazioni, si calcola la matrice di varianza e covarianza;
- il tempo di lavorazione (r_i nel modello) per ogni prodotto è un numero estratto da una distribuzione uniforme tra 1 e 5;
- la capacità disponibile in termine di tempo è stata calcolata facendo il prodotto scalare tra il vettore dei valori attesi delle domande e quello dei tempi di lavorazione e moltiplicando il risultato per un fattore di capacità che si è scelto essere pari a 2 (per non avere un vincolo troppo 'esigente');
- il tempo di setup ($r_i^{'}$ nel modello) si calcola alla base di un fattore di tempo, che si è scelto essere pari a 0.5: questo fattore significa che, se ad un istante di tempo fosse necessario il setup per tutti i prodotti, il 50 % della capacità disponibile sarebbe impiegata per i setup. A partire da questo tempo 'totale' per tutti i setup, si è supposto per semplicità che tutti i prodotti avessero lo stesso tempo di setup;
- il livello iniziale di magazzino per un prodotto è un numero intero estratto in modo uniforme tra 0 ed il doppio del valore atteso della domanda per il prodotto in questione;
- si suppone che il numero di prodotti in questione sia 5.

Le combinazioni di (h_fact, g_fact, TBO) generano 12 modelli distinti, ognuno dei quali è stato risolto 10 volte usando 10 diversi seed per l'albero di scenari, sia con il solver standard che con il solver Fix & Relax. Per quanto riguarda il solver standard, si è sempre passato come valore di gap (per l'arresto) l'1%, mentre il limite di tempo varia per ogni albero. Per il solver Fix & Relax invece si è mantenuto il gap di default (0.01 %) e non si è imposto alcun limite di tempo. Si riportano in seguito una serie di grafici (una per ogni fattore di ramificazione esaminato) che confrontano i risultati in sample, sia in termini di tempo che in termini di funzione obiettivo, dei due diversi solver.
I grafici 2.2 e 2.3 confrontano le prestazioni dei due solver, con un albero di scenari caratterizzato da fattori di ramificazione [5,3,2]. Ogni osservazione è la media dei dieci risultati ottenuti usando questo albero (con i diversi seed) sul problema definito dalla configurazione dei fattori corrispondente al valore indicato sull'asse delle ascisse. Dal primo grafico della figura 2.2 si vede che le due linee spezzate sono praticamente sovrapposte, dunque i valori delle funzioni obiettivo sono quasi identici con i due solver. Dal secondo grafico si evince che la differenza relativa tra le funzioni obiettivo è quasi sempre sotto l'1%, dove essa è definita (per un singolo modello) come

$$\frac{(objV\_FR - objV\_S)}{objV\_S},$$

in cui $objV\_S$ indica il valore della funzione obiettivo trovato con il solver standard e $objV\_FR$ il valore della funzione obiettivo trovato con il solver Fix & Relax.

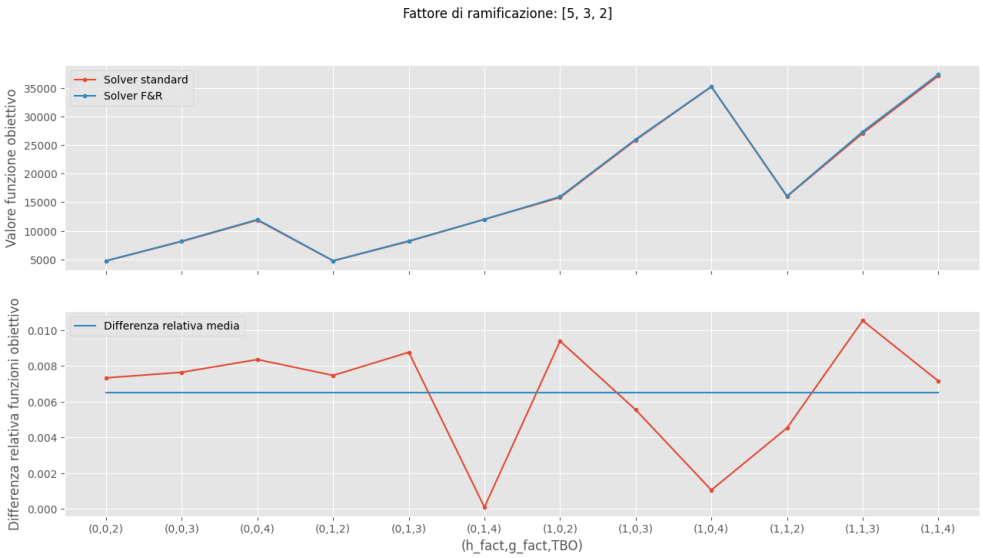

Figura 2.2. Fattore di ramificazione [5,3,2]: valori obiettivo

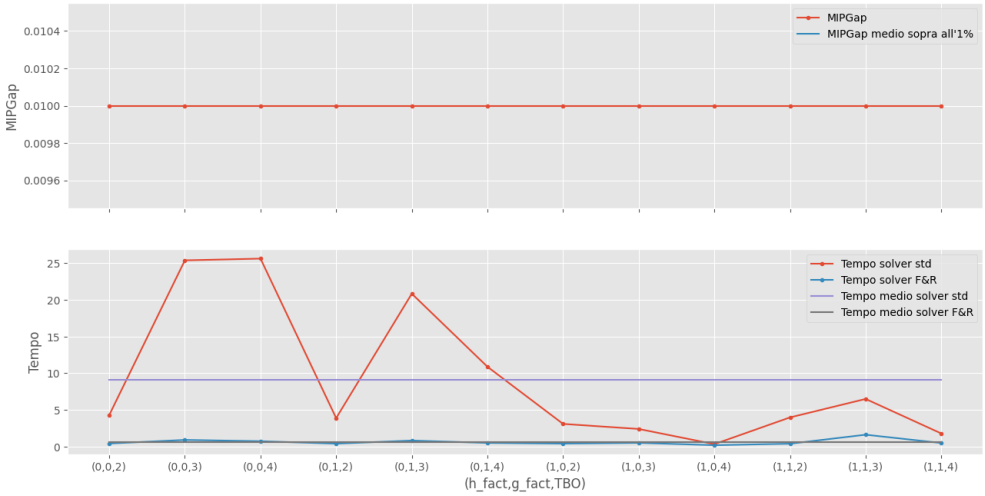

Figura 2.3. Fattore di ramificazione [5,3,2]: gap e tempi

La differenza relativa media (su tutti i 120 problemi risolti con questi fattori di ramificazione) è inferiore allo 0.7%.
Il primo grafico della figura 2.3 rappresenta il gap tra il limite inferiore ed il limite superiore per la soluzione ottimale (nell'algoritmo branch and bound) raggiunto dal solver standard alla soluzione trovata. Se per ognuno dei dieci problemi risolti per ogni configurazione dei fattori la soluzione ha raggiunto il gap passato come input al solver (sempre l'1%, anche con i successivi fattori di ramificazione), sul grafico è riportato il valore 0.01. Se invece qualcuno di essi non raggiunge il gap richiesto (dunque la fase di risoluzione si arresta per aver raggiunto il limite massimo di tempo imposto), il valore riportato sul grafico è la media tra i gap raggiunti alla soluzione, limitata ai problemi in cui non è stato raggiunto l'1%. In questo caso vediamo che il solver standard raggiunge sempre il gap dell'1%, dunque la soluzione trovata è accurata (e di conseguenza anche la sua approssimazione trovata con il solver Fix & Relax).
L'ultimo grafico riporta i tempi computazionali richiesti per arrivare alla soluzione usando i due diversi solver, ed i rispettivi tempi medi (su tutte le configurazioni di fattori). Con questo tipo di albero il solver standard è veloce (tempo medio inferiore ai 10 secondi), ma quello Fix & Relax si comporta comunque nettamente meglio (tempo medio inferiore ad 1 secondo).
I seguenti grafici (2.4 e 2.5) illustrano i risultati delle simulazioni fatte con un albero caratterizzato da fattori di ramificazione [6,5,2]. I valori delle funzioni obiettivo con i due solver sono ancora quasi identici e la differenza relativa media tra i valori è nuovamente inferiore allo 0.7%.

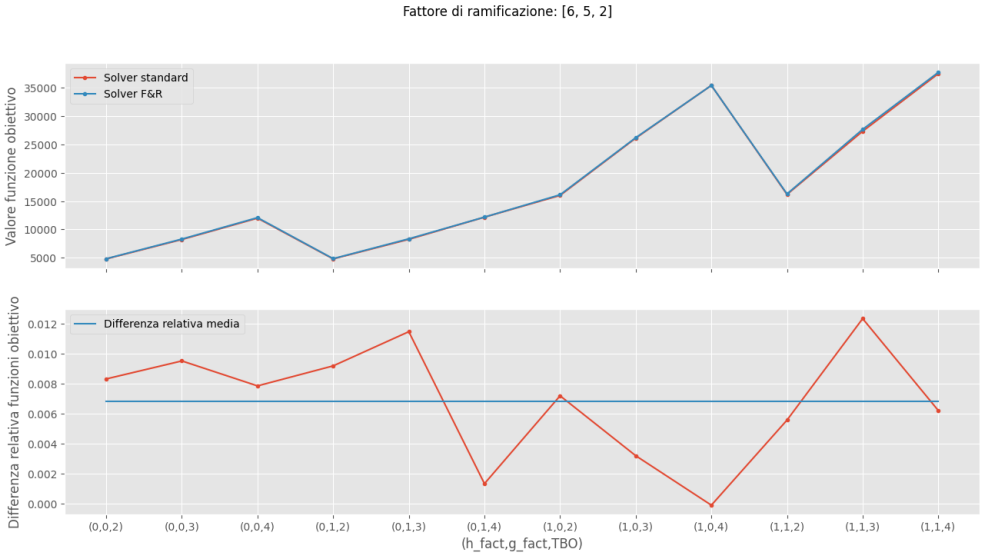

Figura 2.4. Fattore di ramificazione [6,5,2]: valori obiettivo

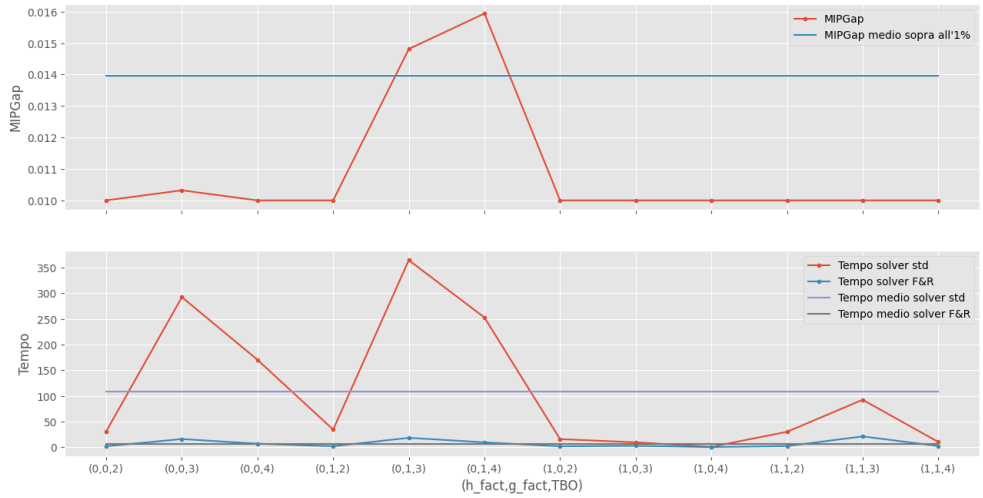

Figura 2.5. Fattore di ramificazione [6,5,2]: gap e tempi

Dal terzo grafico notiamo come in questo caso talvolta il solver standard si arresti per aver raggiunto il limite di tempo (per questo albero 900 secondi). Questo avviene poche volte, solo 4 sul totale delle 120 simulazioni, ed il gap medio raggiunto in questi casi è

circa 1.4%, dunque la soluzione è comunque accurata. Per quanto riguarda i tempi medi, in questo caso la differenza risulta molto più evidente: quello del solver standard è 108 secondi (quasi 2 minuti), quello del solver Fix & Relax è di appena 7 secondi.
Il terzo albero analizzato è quello caratterizzato da fattori di ramificazione [5,3,2,2], dunque ha un periodo di tempo in più rispetto ai precedenti due.
Dai due grafici della figura 2.6 si vede come i valori della funzione obiettivo trovati dai due diversi solver sono di nuovo quasi identici e che la differenza relativa media tra di essi è di pochissimo superiore all'1%.
Passando alla figura 2.7, anche in questo caso succede che il solver standard non sempre si arresti per aver raggiunto il gap dell'1%, ma per il limite di tempo (anche per questo albero 900 secondi). In questo caso succede per 25 simulazioni sulle 120 totali. Il gap medio che è stato raggiunto in queste 25 simulazioni è molto simile al caso precedente, sempre di poco inferiore all'1.4%. La qualità della soluzione dunque è buona anche in questo caso.
Infine, come si vede dall'ultimo grafico della figura 2.7, il tempo medio impiegato dal solver standard è di poco inferiore ai 300 secondi, mentre quello impiegato dal solver Fix & Relax è di 7 secondi.

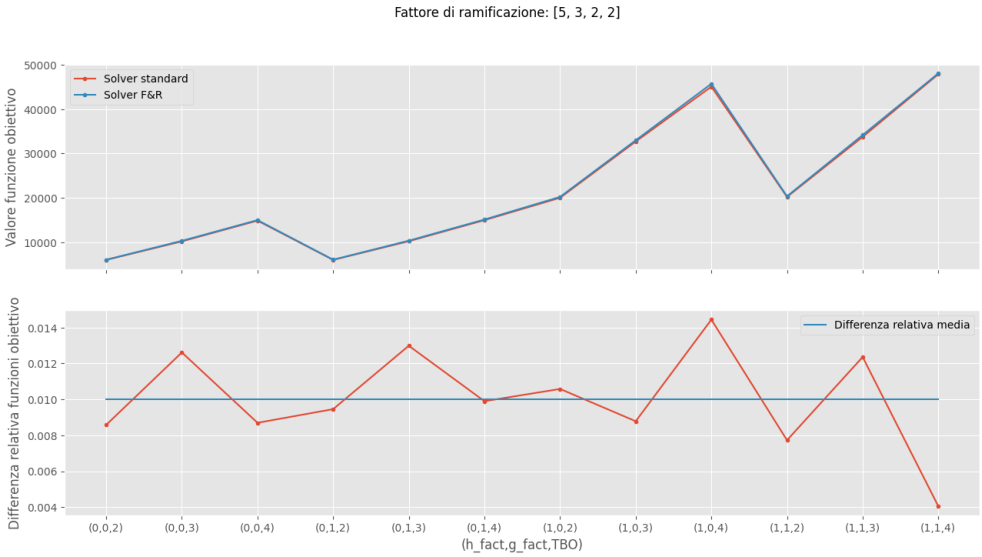

Figura 2.6. Fattore di ramificazione [5,3,2,2]: valori obiettivo

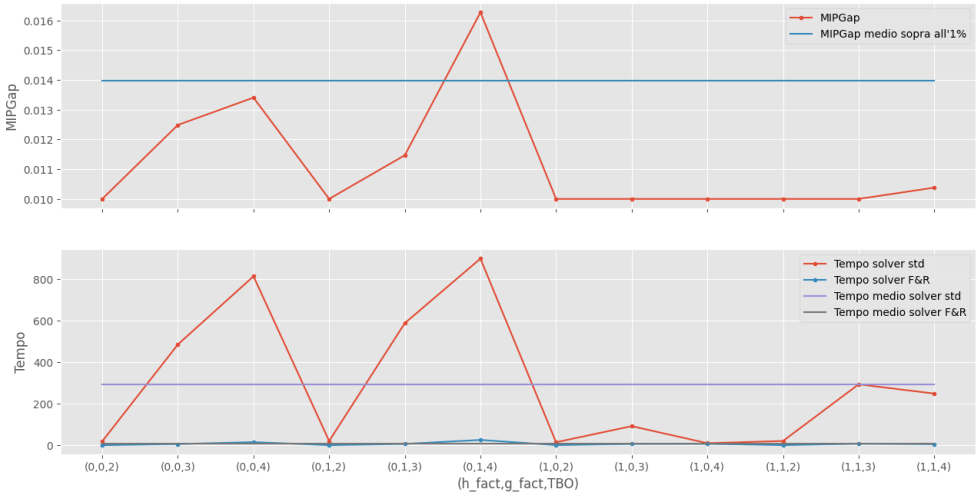

Figura 2.7. Fattore di ramificazione [5,3,2,2]: gap e tempi

Si ripete la stessa analisi grafica per il quarto albero analizzato, con fattori di ramificazione [9,5,2,1]. Ancora una volta i due solver trovano soluzioni con un valore della funzione obiettivo praticamente identico, la differenza relativa media tra essi è inferiore all'1% (grafici della figura 2.8).

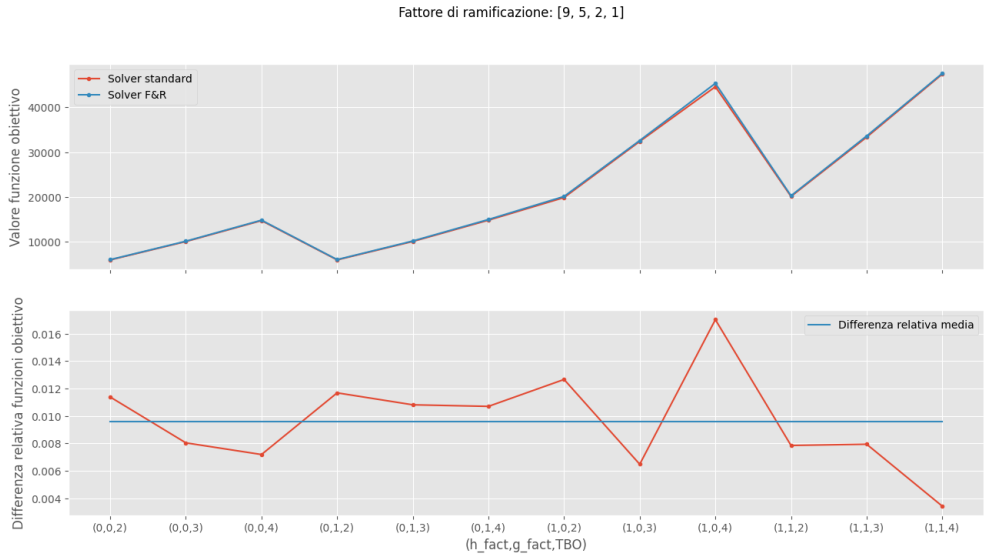

Figura 2.8. Fattore di ramificazione [9,5,2,1]: valori obiettivo

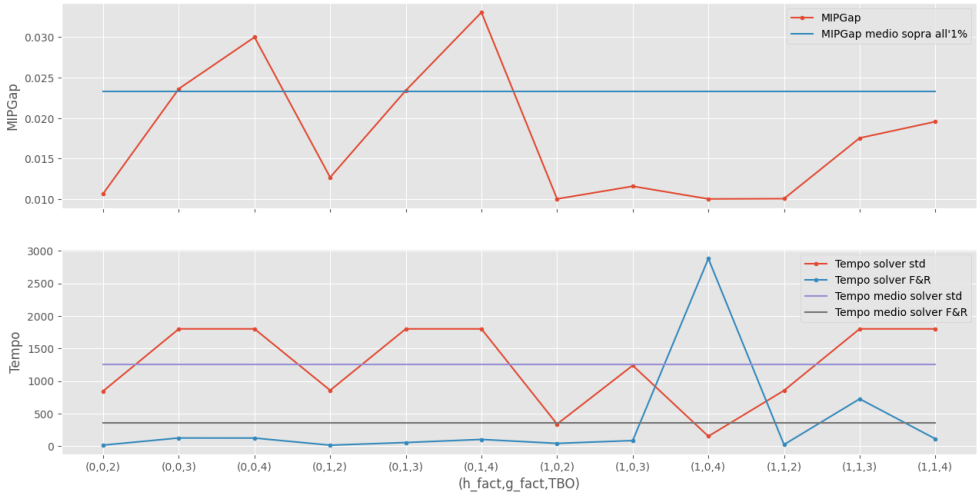

Figura 2.9. Fattore di ramificazione [9,5,2,1]: gap e tempi

Con questo albero più 'complesso' (nel senso che ha un maggior numero di nodi rispetto al precedente, a parità di profondità) aumenta il numero di simulazioni in cui il solver standard si arresta per il limite di tempo (in questo caso 1800 secondi, cioè 30 minuti). Più precisamente, questo accade in 66 simulazioni su 120. Il gap medio (primo grafico della figura 2.9) raggiunto in queste simulazioni è di poco inferiore al 2.4%, superiore rispetto ai casi precedenti, ma ancora tale da poter considerare la soluzione accurata.
Il tempo medio del solver standard è superiore a 1250 secondi, quello del solver che utilizza la mateuristica è superiore ai 380 secondi.
Si nota subito un'anomalia rispetto ai casi precedenti: con questo albero di scenari, per la configurazione $(h\_fact, g\_fact, TBO) = (1,0,4)$ il tempo medio sulle dieci simulazioni del solver Fix & Relax è nettamente superiore rispetto a quello del solver standard. Questo risultato non è però in contraddizione con la teoria, perché i due solver utilizzano due gap differenti. Quello standard si arresta ad un gap dell'1%, mentre quello con la mateuristica si arresta (ad ogni iterazione) allo 0.01%. Per questa configurazione dei fattori, questa maggiore accuratezza richiesta alla soluzione di ognuno dei sotto problemi si traduce in tempi di calcolo esageratamente lunghi.
Si è quindi deciso di ripetere la simulazione per tutti i casi in cui il tempo computazionale del solver Fix & Relax era superiore a quello del solver standard, aumentando il gap all'1% ad ogni iterazione.
Dal secondo grafico della figura 2.11 si nota come ora l'anomalia non sia più presente, ed il tempo computazionale medio del solver che utilizza la mateuristica è diminuito a circa 103 secondi. Il prezzo che si paga è una minore accuratezza dell'approssimazione della soluzione data dalla mateuristica in corrispondenza della configurazione $(h\_fact, g\_fact, TBO) = (1,0,4)$. Dal secondo grafico della figura 2.10 si vede che la sua

differenza relativa supera l'1.8%, mentre nel grafico 2.8 essa era inferiore, circa 1.75%. L'approssimazione della mateuristica resta comunque ancora accurata.

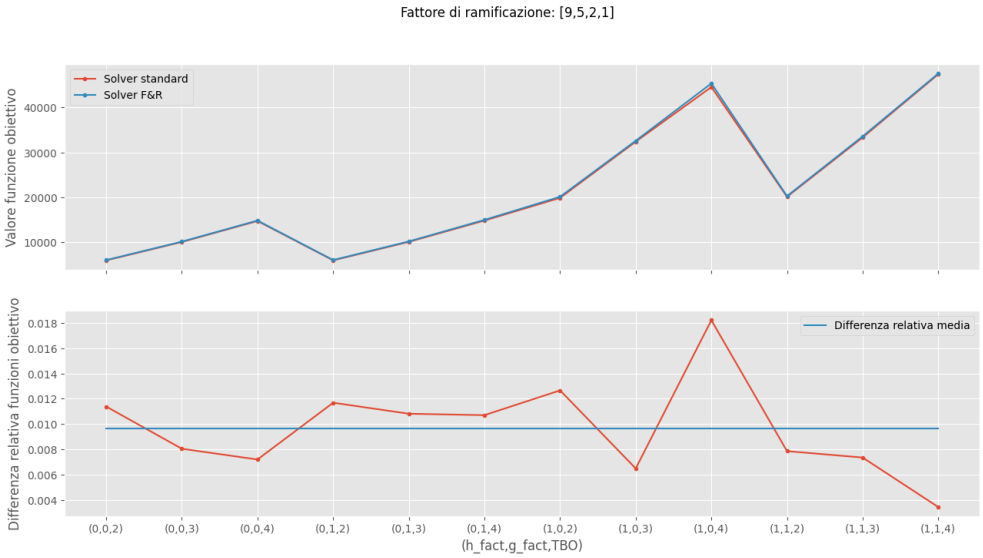

Figura 2.10. Fattore di ramificazione [9,5,2,1]: valori obiettivo dopo l'analisi dell'anomalia

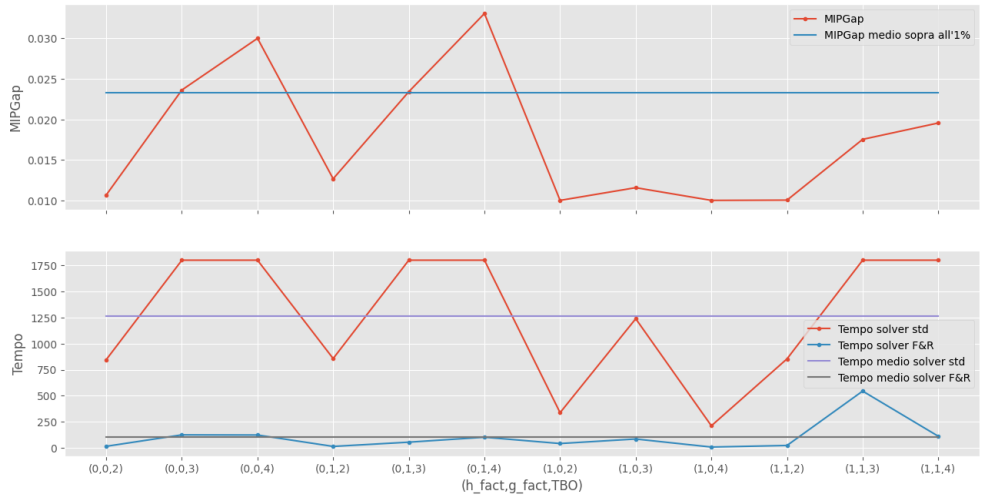

Figura 2.11. Fattore di ramificazione [9,5,2,1]: gap e tempi dopo l'analisi dell'anomalia

L'ultimo albero che si è analizzato è quello di tipo [5,3,3,2,1], dunque con un ulteriore periodo temporale in più rispetto ai due precedenti alberi.
I valori delle funzioni obiettivo sono ancora una volta molto simili, la differenza relativa

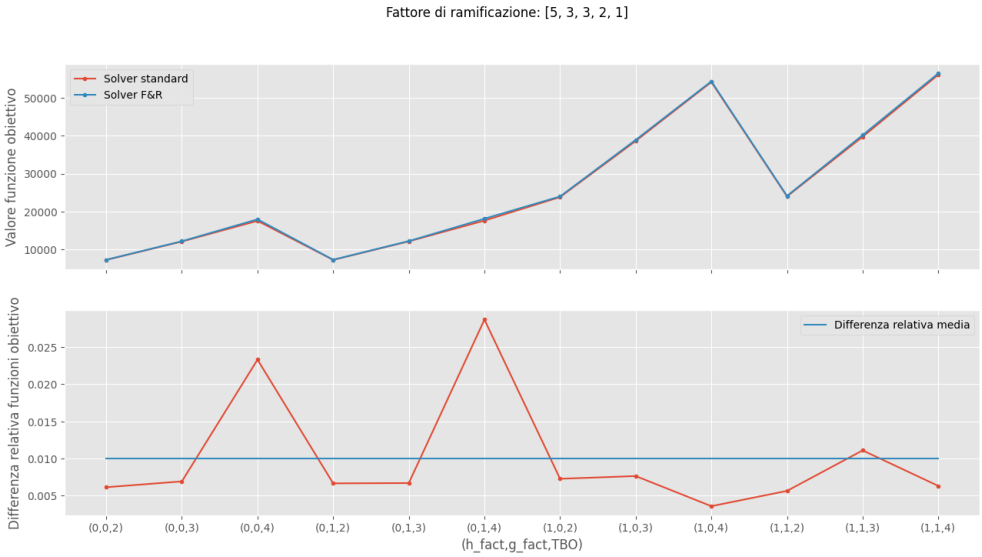

Figura 2.12. Fattore di ramificazione [5,3,3,2,1]: valori obiettivo

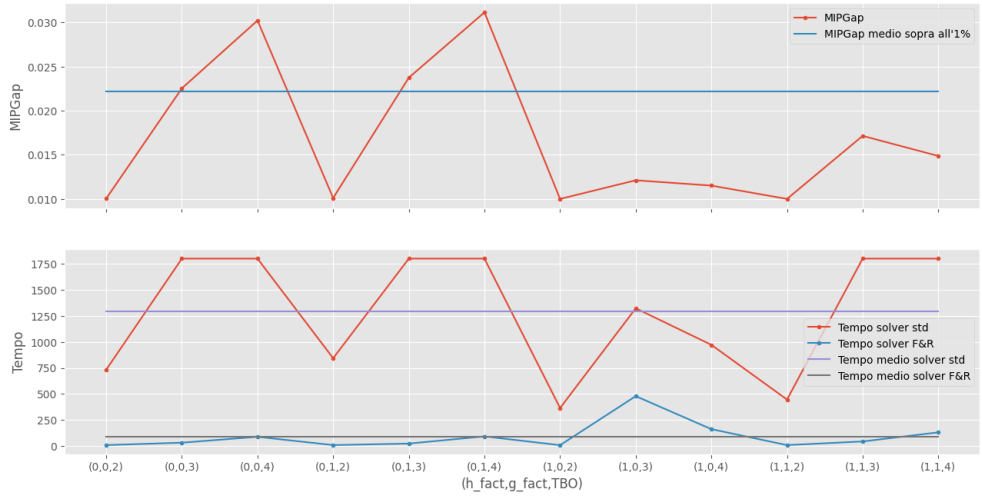

Figura 2.13. Fattore di ramificazione [5,3,3,2,1]: gap e tempi

media tra di essi è dell'1% (figura 2.12). Anche in questo caso, in 66 simulazioni sulle 120 il solver standard si arresta per aver raggiunto il limite di tempo (ancora 1800 secondi) e non il gap dell'1%. Il gap medio raggiunto in questi casi è del 2.2%, inferiore a quanto trovato per l'albero di tipo [9,5,2,1]. Il tempo computazionale medio è di 1290 secondi per il solver standard, di 90 secondi per il solver che utilizza la mateuristica.
Si conclude questa sezione con i grafici precedenti fatti su tutte le simulazioni, non più divise per albero, e con una breve analisi statistica del comportamento dei solver rispetto ai fattori presi in esame (costi di magazzino bassi o alti, penalità per vendite perse basse o alte, tempo medio tra ordini).
La differenza relativa media tra i valori delle funzioni obiettivo è dello 0.85%, dunque estremamente bassa. Il gap medio raggiunto, nei casi in cui il solver standard si arresta per il limite di tempo (che sono 161, sulle 600 simulazioni complessive), è 2.1%. Infine i tempi computazionali medi (sul totale delle simulazioni) sono 591 secondi per il solver standard, 90 secondi per quello Fix & Relax (nonostante il lungo tempo di risoluzione richiesto per la configurazione $(h\_fact, g\_fact, TBO) = (1,0,4)$ con l'albero [9,5,2,1]).
Possiamo quindi concludere che la mateuristica approssima bene il valore della funzione obiettivo trovato dal solver standard, che a sua volta raggiunge sempre un gap abbastanza ridotto dalla soluzione ottima. I tempi computazionali sono decisamente inferiori, dunque nel proseguimento, in cui si andranno a effettuare delle simulazioni di tipo rolling horizon, si utilizzerà il solver Fix & Relax.

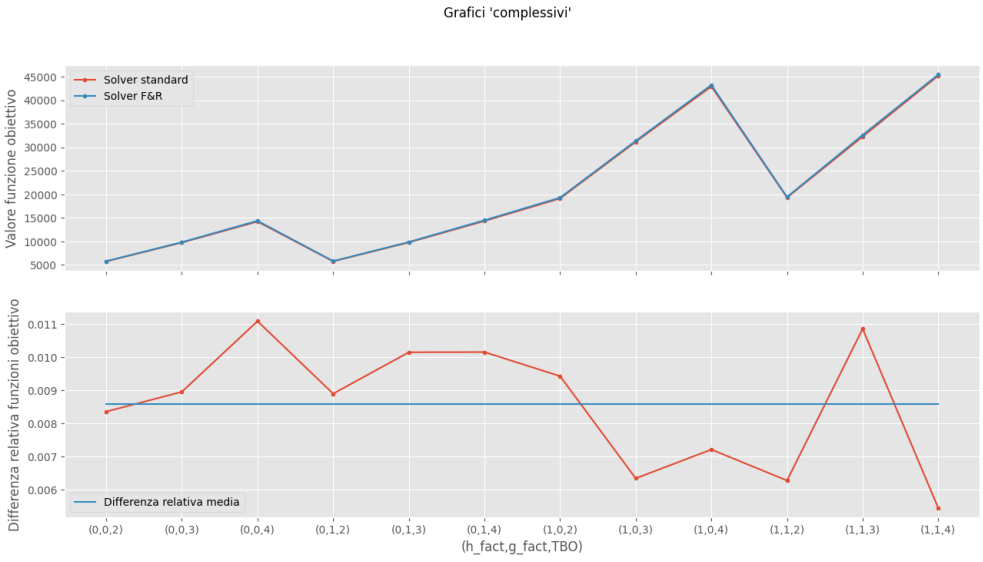

Figura 2.14. Tutti i fattori di ramificazione: valori obiettivo

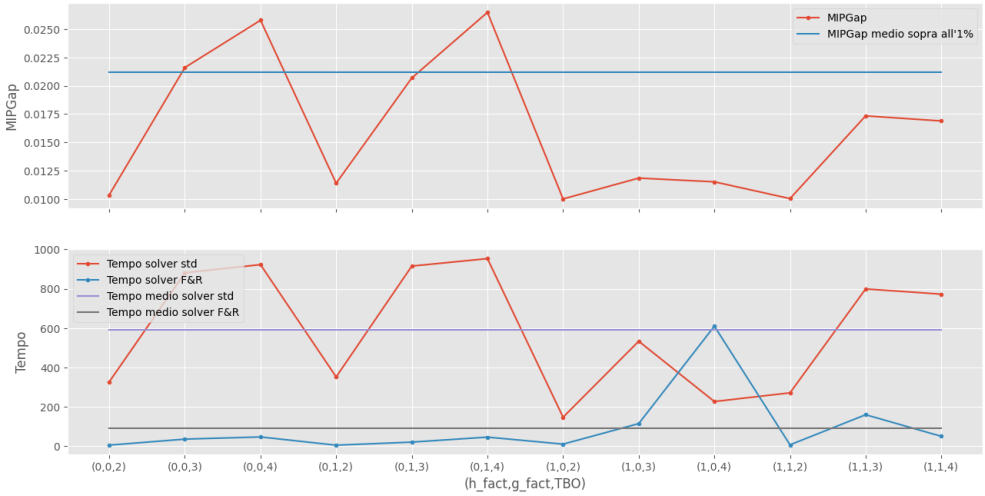

Figura 2.15. Tutti i fattori di ramificazione: gap e tempi

Infine, si è effettuata una analisi della varianza (ANOVA) per valutare l'effetto dei fattori (costi di magazzino, penalità domande perse e tempo medio tra ordini) sul gap raggiunto dal solver standard e sul suo tempo computazionale.
Si riporta in seguito il risultato dell'ANOVA a tre vie (i tre fattori sopra citati, con anche le interazioni) fatta rispetto al tempo computazionale del solver standard. L'ipotesi nulla è che le differenze osservate tra i gruppi (determinati dai valori dei fattori) dipendano dal caso.

	sum_sq	df	F	PR(>F)
h_fact	1.066526e+07	1.0	23.639789	1.490280e−06
g_fact	4.372328e+06	1.0	9.691367	1.940703e−03
TBO	1.974020e+07	1.0	43.754610	8.334337e−11
h_fact : g_fact	2.976809e+06	1.0	6.598168	1.045232e−02
h_fact :TBO	2.365665e+06	1.0	5.243552	2.237917e−02
g_fact :TBO	1.142088e+06	1.0	2.531466	1.121303e−01
h_fact : g_fact :TBO	1.083373e+06	1.0	2.401321	1.217673e−01
Residual	2.670849e+08	592.0	NaN	NaN

L'ipotesi nulla si può sicuramente rifiutare per tutti e tre i singoli fattori, come si può notare dall'ultima colonna. La probabilità di osservare un valore della statistica F maggiore di quello trovato, se fosse vera l'ipotesi nulla, è troppo bassa. Per quanto riguarda le interazioni, l'ipotesi nulla non si può sicuramente rifiutare nel caso dell'interazione tra tutti e tre i fattori e nel caso di interazione g_fact con TBO: il valore della probabilità è troppo alto. Anche per le altre due interazioni, pur avendo valori di statistica F più grandi e dunque probabilità dell'ultima colonna inferiore, non si può rifiutare facilmente l'ipotesi nulla, dipende dal livello di significatività scelto.

Si riportano in seguito i coefficienti (solo dei tre fattori) calcolati dalla regressione lineare relativa allo stesso modello (variabile dipendente il tempo computazionale del solver standard, variabili indipendenti i tre fattori e le loro interazioni).

	coef	std err	t	P>|t|
Intercept	−183.0288	208.834	−0.876	0.381
h_fact	366.1544	295.337	1.240	0.216
g_fact	21.5066	295.337	0.073	0.942
TBO	297.6616	67.168	4.432	0.000

Si vede che l'unico coefficiente 'significativo' è quello relativo al TBO, e come ci si aspettava (anche visivamente dal secondo grafico della figura 2.15) esso è maggiore di zero, dunque tendenzialmente all'aumentare del tempo medio tra ordini aumenta il tempo computazionale del solver standard.
Si è ripetuta la stessa analisi per vedere l'effetto dei fattori sul gap raggiunto dal solver standard.

	sum_sq	df	F	PR(>F)
h_fact	1.526871e−03	1.0	37.509727	7.349122e−09
g_fact	2.497770e−05	1.0	0.613612	4.346418e−01
TBO	9.612582e−04	1.0	23.614663	2.887850e−06
h_fact : g_fact	1.261969e−04	1.0	3.100204	8.028039e−02
h_fact : TBO	2.863145e−04	1.0	7.033719	8.841932e−03
g_fact : TBO	8.226206e−06	1.0	0.202088	6.536767e−01
h_fact : g_fact : TBO	1.307696e−08	1.0	0.000321	9.857232e−01
Residual	6.228017e−03	153.0	NaN	NaN

L'ipotesi nulla dell'ANOVA si può rifiutare per i fattori del costo di magazzino e del TBO, ed anche per l'interazione tra quei due fattori. Negli altri casi è difficile rifiutare l'ipotesi nulla, i valori della statistica F sono tutti bassi.

	coef	std err	t	P>|t|
Intercept	0.0045	0.005	0.826	0.410
h_fact	0.0083	0.022	0.370	0.712
g_fact	−0.0032	0.008	−0.428	0.669
TBO	0.0054	0.002	3.493	0.001

Anche questa volta, si vede che l'unico dei coefficienti della regressione lineare associata al modello ad essere significativo è quello del TBO. Esso è nuovamente positivo (con intervallo di confidenza interamente maggiore di zero), dunque tendenzialmente il gap raggiunto dal solver standard è maggiore per TBO più grandi, ovvero la soluzione trovata è più distante da quella ottima.

2.4 Risultati out of sample del solver F&R

Dopo aver verificato il comportamento in sample del solver, in questa sezione lo si sperimenta su scenari di domanda out of sample. Il tipo di simulazione scelto, come anticipato nell'introduzione, è il rolling horizon. Il seguente listato di codice mostra i passaggi chiave di questo tipo di simulazione.

```
for t in range(horizon):

    #DECISIONE DI PRODUZIONE Y
    [objV,Y,S,comp_time]=solver.solve(instance1, Tree1, I)

    #AGGIORNAMENTO MAGAZZINO
    for k in range(num_items):
        if I[k]+Y[k]-D[path,t,k]>0:
            I[k]=I[k]+Y[k]-D[path,t,k]
            Z[k]=0
        else:
            Z[k]=-(I[k]+Y[k]-D[path,t,k])
            I[k]=0

    #COSTO DI MAGAZZINO
    h_cost=np.inner(I,h)
    #COSTO PER LA DOMANDA NON SODDISFATTA
    g_cost=np.inner(Z,g)
    #COSTI DI SETUP
    f_cost=np.inner(S,f)
    #COSTO COMPLESSIVO ALL'ISTANTE t
    cost=f_cost+h_cost+g_cost
    #AGGIORNAMENTO COSTO E TEMPO COMPLESSIVO
    cum_cost+=cost
    cum_time+=comp_time
```

Dato un percorso di *horizon* istanti temporali ed una condizione iniziale di magazzino, ad ogni istante si osserva il livello di magazzino e si usa il solver per decidere la quantità da produrre di ogni oggetto e le relative variabili di set up. Dopo aver prodotto si scopre la domanda e, se possibile, la si soddisfa, aggiornando il magazzino e la eventuale domanda non soddisfatta (i vettori I e Z nel codice). Infine si calcolano i tre costi, facendo i prodotti scalari tra i vettori delle variabili I, Z e S ed i vettori dei costi h, g e f, la loro somma dà il costo complessivo all'iterazione t. I valori *cum_cost* e *cum_time* servono per poi calcolare il totale dei costi e del tempo computazionale sul percorso di domanda.
Si è utilizzato il solver Fix & Relax, perché come si è visto nella sezione precedente è molto più veloce di quello standard e l'approssimazione della soluzione è buona. Come valore di gap di arresto si è lasciato quello standard, ovvero lo 0.01%, mentre non si è imposto un tempo limite massimo per la risoluzione dei problemi.

Si riportano in seguito le ipotesi fatte sui parametri del problema per le simulazioni, i cui risultati verranno in seguito esposti:

- si suppone che il numero di prodotti sia 5;
- il valore atteso della domanda per ogni prodotto è un numero intero estratto in maniera uniforme tra 10 e 100. Il coefficiente di variazione per ogni prodotto è estratto in maniera uniforme tra 0.1 e 0.5. A partire da questi valori si calcolano le varianze delle domande. Per generare la matrice di varianza-covarianza, si estrae una matrice di correlazione (con elementi pari ad 1 sulla diagonale e con elementi extra diagonali estratti in maniera uniforme tra -1 ed 1, in modo che la matrice sia simmetrica). Date le varianze e le correlazioni, si calcola la matrice di varianza e covarianza;
- il livello iniziale di magazzino per un prodotto è un numero intero estratto in modo uniforme tra 0 ed il doppio del valore atteso della domanda per il prodotto in questione;
- il costo di magazzino di ogni prodotto è un numero intero estratto in maniera uniforme tra 1 e 10;
- la penalità per la domanda non soddisfatta è un numero intero estratto in maniera uniforme tra 50 e 100;
- costi di set up sono calcolati secondo la formula $f_i = \frac{D_i}{2} T_{bo,i}^2 h_i$, in cui si è scelto di usare il valore di $T_{bo} = 2$ per ogni prodotto, dato che si è visto nella sezione precedente che all'aumentare di questo parametro aumenta la difficoltà computazionale del problema;
- il tempo di lavorazione (r_i nel modello) per ogni prodotto è un numero estratto da una distribuzione uniforme tra 1 e 5;
- la capacità disponibile in termine di tempo è stata calcolata facendo il prodotto scalare tra il vettore dei valori attesi delle domande e quello dei tempi di lavorazione e moltiplicando il risultato per un fattore di capacità che si è scelto essere pari a 2 (per non avere un vincolo troppo 'esigente');
- il tempo di setup ($r_i^{'}$ nel modello) si calcola a partire da un fattore di tempo, che si è scelto essere pari a 0.5: questo fattore significa che se ad un istante di tempo fosse necessario il setup per tutti i prodotti, il 50 % della capacità disponibile sarebbe impiegata per i setup. A partire da questo tempo 'totale' per tutti i setup, si è supposto per semplicità che tutti i prodotti avessero lo stesso tempo di setup.

Per quanto riguarda le domande, si sono estratti 5 percorsi di domande, ognuno della durata di 50 istanti temporali. Per ogni istante temporale è stato estratto un campione da una distribuzione normale multivariata con il vettore delle medie e la matrice di varianza-covarianza estratti. Nel caso in cui qualche componente estratta fosse negativa (possibile in una normale multivariata, ma privo di senso per una domanda), esse vengono poste uguali

a zero per ipotesi. Questi valori (vettore delle medie, matrice di varianza e covarianza e percorsi di domanda out of sample) sono rimasti gli stessi in tutte le simulazioni effettuate. Per quanto riguarda invece gli altri parametri esposti in precedenza, sono stati estratti con 10 seed differenti, in modo da vedere il comportamento del solver simulando su diverse configurazioni.
Si riportano nel seguito alcuni grafici che mostrano i risultati di queste simulazioni. La prima figura (2.16) studia il comportamento del costo lungo i 50 istanti temporali dei percorsi di domanda. I valori sull'asse delle ascisse sono gli istanti temporali (indicizzati da 0 a 49, in modo concorde a quanto fatto su Python), quelli sull'asse delle ordinate sono le 'medie mobili' del costo. Dato un percorso di domande su cui sono state effettuate le simulazioni e calcolati i costi istante per istante, con 'media mobile' del costo all'istante t (con t che va da 0 a 49) si intende la quantità

$$\overline{Costo}(t) = \frac{CumCost(t)}{t+1} \tag{2.8}$$

dove

$$CumCost(t) = \sum_{i=0}^{t} Cost(i)$$

La media è mobile nel senso che ad ogni istante di tempo si sommano tutti costi dei singoli periodi dal primo a quello attuale e si divide per il numero di periodi in questione.

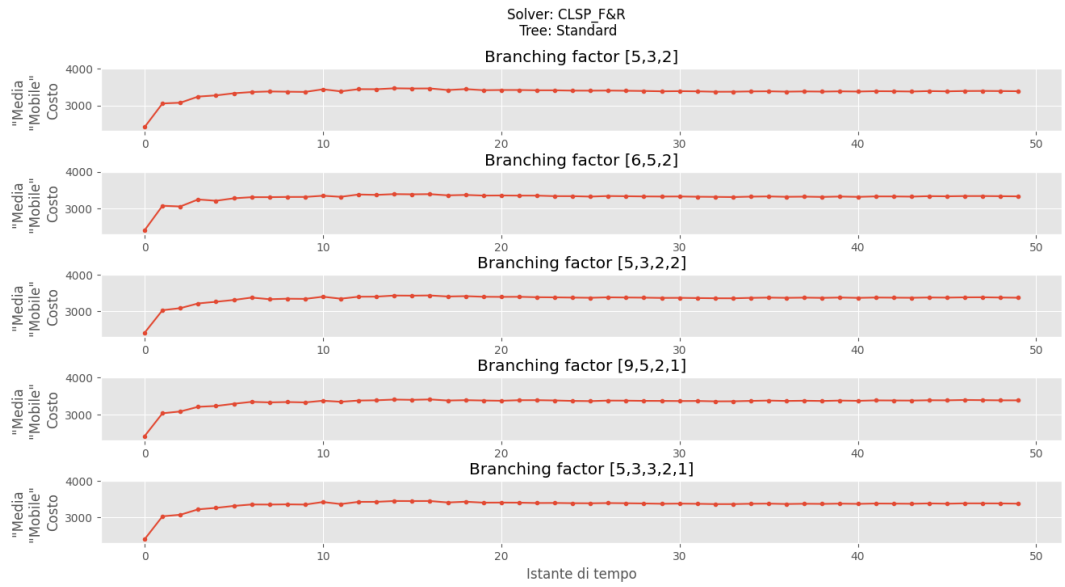

Figura 2.16. Media mobile del costo

Nella figura sono presenti 5 grafici, uno per ciascuno dei fattori di ramificazione presi in esame (gli stessi della sezione precedente). Ogni singolo punto nei grafici è la media di 50 medie mobili, una per ogni coppia di seed per l'estrazione dei parametri (10 valori) e percorso di domanda (5 percorsi).
L'andamento dei 5 grafici è molto simile. In ognuno di essi, dopo un transiente iniziale, la media mobile del costo si assesta attorno ad un valore che varia molto poco fino alla fine dell'orizzonte temporale. Questo è coerente con ciò che ci si aspetta: la fase iniziale è influenzata dal livello iniziale di magazzino, dunque i costi possono variare abbastanza di istante in istante. Dopo una decina di periodi invece il costo medio mobile si stabilizza, dunque la spesa è quasi sempre la stessa. Le decisioni prese dal solver sembrano essere quindi buone, perché portano ad un costo costante nel tempo e non soggetto a sbalzi importanti.
La figura 2.17 e la tabella 2.1 mettono a confronto i risultati delle simulazioni, sia in termini di costo complessivo sull'orizzonte temporale che in termini di tempi di calcolo, dividendole in base al fattore di ramificazione dell'albero che si passa in ingresso al solver.

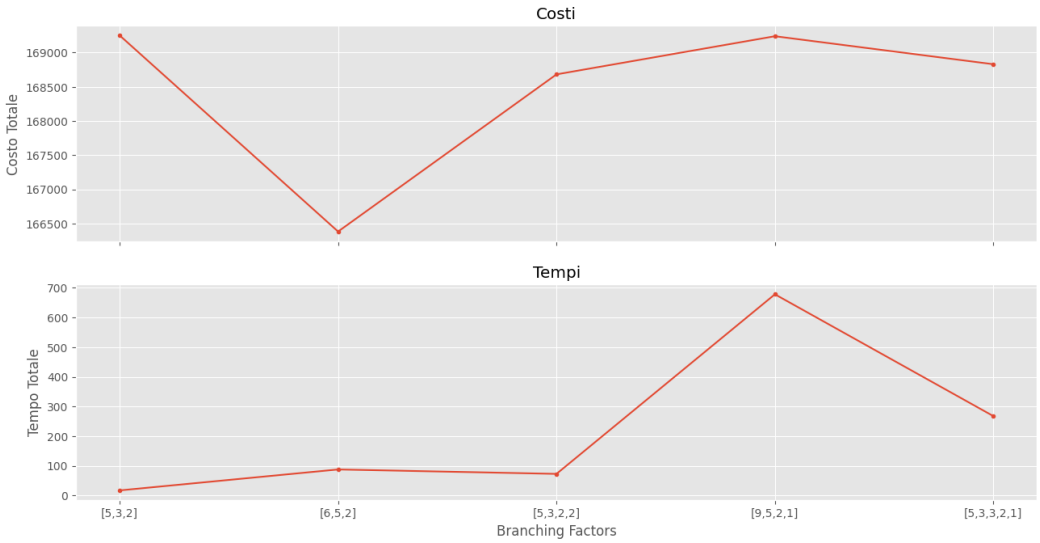

Figura 2.17. Costi e tempi totali

Fattore di ramificazione	*Costo totale*	*Tempo totale*
[5,3,2]	169248.90210629985	17.054624438285828
[6,5,2]	166383.37822730807	87.6777483844757
[5,3,2,2]	168679.9936461453	72.77375486373901
[9,5,2,1]	169237.4710563179	678.3568441772461
[5,3,3,2,1]	168828.4203705439	267.72785397052763

Tabella 2.1. Valori numerici della figura 2.17

Ogni punto dei grafici è la media dei risultati (del costo o del tempo complessivo) di 50 simulazioni, una per ogni coppia di seed dei parametri (10 totali) e percorsi di domanda (5 totali). Dal primo grafico della figura si vede che il fattore di ramificazione con cui si minimizza il costo totale è il [6,5,2], mentre gli altri ottengono risultati di costi simili. Per quanto riguarda i tempi computazionali, gli ultimi due fattori impiegano tempi molto grandi rispetto ai primi tre (ovviamente, essendo più profondi e con più nodi, danno vita a modelli più complicati da risolvere). I loro risultati in termini di costi però sembrano suggerire che non sia conveniente utilizzare un albero troppo complicato per questo tipo di problema.
Si riporta ancora un'ultima figura che mostra il comportamento dei costi, divisi in base al loro tipo (magazzino, penalità per non aver soddisfatto la domanda o setup).

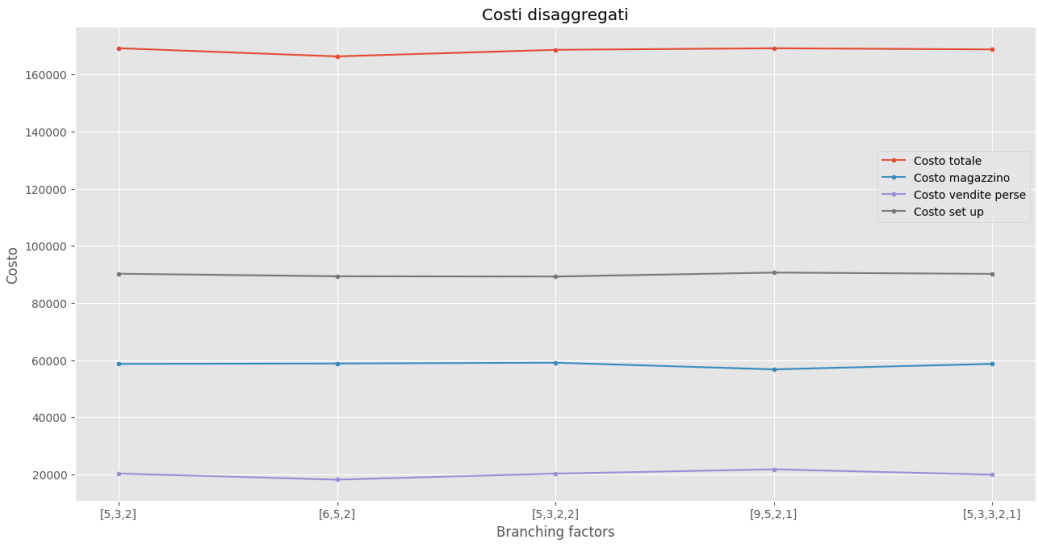

Figura 2.18. Costi totali disaggregati

Dalla figura 2.18 e dalla tabella 2.2 si vede che per tutti i fattori di ramificazione il

Fattore	*Costo totale*	*Costo magazzino*	*Costo vendite perse*	*Costo setup*
[5,3,2]	169248.90210629985	58693.06340421281	20283.838702087058	90272.0
[6,5,2]	166383.37822730807	58839.90515618728	18162.353071120833	89381.12
[5,3,2,2]	168679.9936461453	59120.2725150468	20266.921131098516	89292.8
[9,5,2,1]	169237.4710563179	56781.1455998794	21777.60545643852	90678.72
[5,3,3,2,1]	168828.4203705439	58706.21084029313	19899.96953025077	90222.24

Tabella 2.2. Valori numerici della figura 2.18

comportamento dei costi disaggregati è simile. Il costo maggiore è quello per i set up (più della metà del costo totale), i fattori che minimizzano questo costo sono il [5,3,2,2] ed il [6,5,2]. I costi di magazzino sono quelli intermedi, il fattore che li minimizza è il [9,5,2,1]. I costi minori sono quelli per le domande non soddisfatte, il fattore di ramificazione che minimizza questo costo è il [6,5,2].

2.5 Riduzione della varianza: albero di scenari "di Sobol"

Per tutte le simulazioni effettuate fino a questo punto, i cui risultati sono stati esposti nel capitolo precedente, l'incertezza della domanda è stata rappresentata attraverso un albero di scenari (passato in ingresso al solver). Come descritto in precedenza, per ogni nodo di questo albero si campiona da una normale multivariata utilizzando la funzione 'random.multivariate_normal' della libreria numpy di Python. Ci si chiede se sia possibile migliorare la qualità dell'albero di scenari (al fine di migliorare le decisioni prese dal solver e di conseguenza ottenere dei risultati migliori nelle simulazioni), utilizzando delle tecniche di riduzione della varianza quando si va a campionare, invece di utilizzare il metodo 'standard'. Tra i tanti metodi di riduzione della varianza, si è scelto di utilizzare delle sequenze a bassa discrepanza, in particolare quelle di tipo Sobol.
La seguente trattazione delle sequenze di Sobol fa riferimento a quanto fatto in Brandimarte [2006a].
Consideriamo l'ipercubo unitario n-dimensionale $I = [0,1]^n$ e supponiamo di avere una sequenza di N punti al suo interno $X^1, X^2, ..., X^N$. L'idea alla base delle sequenze a bassa discrepanza è che, se i punti sono ben distribuiti all'interno dell'ipercubo, prendendo qualsiasi suo sottoinsieme il numero di punti all'interno del sottoinsieme dovrebbe essere proporzionale al suo volume (essendo $vol(I) = 1$).
Per poter valutare quanto bene i punti di una sequenza siano distribuiti, occorre definire una misura di discrepanza. Considerare qualsiasi sottoinsieme dell'ipercubo sarebbe troppo dispendioso, per semplicità ci si riduce a quelli definiti nel modo seguente. Dato un punto $x = (x_1, x_2, ..., x_n) \in I$, definiamo il sottoinsieme rettangolare

$$G_x = [0, x_1) \times [0, x_2) \times ... \times [0, x_n)$$

che è ovviamente contenuto nell'ipercubo I ed ha volume $vol(G_x) = x_1x_2...x_n$. Sia $S_N(A)$ la funzione che conta il numero di punti della sequenza $X^1, X^2, ..., X^N$ presenti all'interno del sottoinsieme A dell'ipercubo unitario.
Allora, una possibile misura di discrepanza è la seguente:

$$D(X^1, X^2, ..., X^N) = \sup_{x \in I} |S_N(G_x) - Nx_1x_2...x_n|$$

La quantità all'interno del valore assoluto è la differenza tra il numero di punti effettivamente presenti nel sottoinsieme (contati dalla funzione S_N) e la frazione di punti che dovrebbero essere presenti se valesse l'idea alla base descritta in precedenza (ovvero il volume del sottoinsieme per il numero di punti della sequenza). Come misura di discrepanza si prende l'estremo superiore di questa quantità, al variare dei punti x all'interno dell'ipercubo. Bisogna quindi cercare sequenze che minimizzino questa misura (o altre, non è ovviamente l'unica possibile), appunto le sequenze a bassa discrepanza.
Un tipo di sequenza che è alla base di molte sequenze a bassa discrepanza è quella di Van der Corput. Dato un numero primo b ed un intero n, l'n-esimo punto della sequenza si calcola nel modo seguente:

- si rappresenta il numero intero n nella base b

$$n = (d_m...d_3d_2d_1d_0)_b = \sum_{k=0}^{m} d_k b^k;$$

- si riflettono le cifre rispetto al punto decimale aggiungendo uno 0, in modo da ottenere un numero all'interno dell'intervallo unitario

$$h(n, b) = (0.d_0d_1d_2d_3...d_m)_b = \sum_{k=0}^{m} d_k b^{-(k+1)}$$

Ovviamente tutti i numeri $d_0, d_1, ...d_m$ sono interi compresi tra 0 e $b - 1$. La seguente tabella mostra i primi dieci numeri della sequenza di Van der Corput con base $b = 2$, svolgendo i due passaggi illustrati sopra.
I numeri della prima e dell'ultima colonna sono ovviamente scritti in base 10. É immediato notare che (non solo in questi primi dieci numeri) la sequenza di Van der Corput che utilizza come base $b = 2$ è l'unica che non contiene numeri consecutivi crescenti, il che è una caratteristica positiva nell'ottica di avere dei punti 'ben distribuiti' nell'ipercubo (senza entrare troppo nel dettaglio, basi grandi implicano tanti numeri consecutivi crescenti e dunque tanti punti disposti su un iperpiano (n-1) dimensionale, se n è la dimensione dell'ipercubo; ad esempio, nel caso di ipercubo bidimensionale e coordinate dei punti generate da due sequenze di Van der Corput con base grande, è abbastanza intuitivo capire che ci sono molte sequenze, anche lunghe, di punti consecutivi che si dispongono su rette parallele, dunque la sequenza non è ben distribuita nello spazio).
Molte sequenze a bassa discrepanza n-dimensionali sono costruite associando ad ogni dimensione dello spazio un numero primo (diverso) e generando la corrispondente sequenza di Van der Corput con quella base. Le sequenze di Sobol invece utilizzano solamente la

n	$(n)_2$	$(h(n,2))_2$	$h(n,2)$
0	0	0	0
1	1	0.1	0.5
2	10	0.01	0.25
3	11	0.11	0.75
4	100	0.001	0.125
5	101	0.101	0.625
6	110	0.011	0.375
7	111	0.111	0.875
8	1000	0.0001	0.0625
9	1001	0.1001	0.5625

Tabella 2.3. Primi dieci numeri della sequenza di Van der Corput con base 2

base 2 per tutte le dimensioni, associandole un meccanismo di permutazione collegato ai polinomi in aritmetica binaria.
Utilizzando le sequenze di Sobol si ottengono dunque sequenze della dimensione desiderata (nel caso del problema del lot sizing, il numero di prodotti) di punti nell'ipercubo unitario. Avendo supposto che la domanda del problema fosse rappresentata da una distribuzione normale multivariata, è abbastanza semplice passare da questi punti a punti che rappresentino suoi campioni.
Consideriamo una normale multivariata n-dimensionale con vettore dei valori attesi μ e matrice di varianza e covarianza Σ. Quest'ultima è simmetrica e definita positiva, è dunque possibile calcolare la sua fattorizzazione di Cholesky $\Sigma = U^T U$, dove U è una matrice triangolare superiore. Avendo a disposizione il fattore di Cholesky U, per generare campioni $X \sim N(\mu, \Sigma)$ si può procedere nel modo seguente:

- generare n campioni indipendenti da una distribuzione normale standard

$$Z_1, Z_2, ..., Z_n \sim N(0,1)$$

- se $Z = [Z_1, Z_2, ..., Z_n]^T$, restituire il valore

$$X = \mu + U^T Z \tag{2.9}$$

La moltiplicazione del vettore delle normali standard indipendenti per il fattore di Cholesky U "assegna" le giuste correlazioni tra le variabili, rendendo l'uscita dell'algoritmo X distribuita come una normale multivariata con vettore dei valori attesi μ e matrice di varianza e covarianza Σ.
Partendo da una sequenza di Sobol, l'algoritmo completo per ottenere un campione della domanda normale multivariata è il seguente:

- estrarre un punto S dalla sequenza di Sobol: ogni sua componente è compresa tra 0 ed 1;

- generare il vettore Z, applicando componente per componente al vettore S la funzione quantile della normale standard;

- calcolare e restituire il campione X applicando la formula 2.9.

Si è definita una seconda classe di alberi, ScenarioTree_Sobol, che genera un albero di scenari che in ogni nodo campiona a partire da una sequenza di Sobol (come nel caso precedente, nel nodo radice e nei nodi figli di un fattore di ramificazione pari ad 1 si osserva il valore atteso della domanda per ipotesi). Si riporta in seguito un listato del codice che mette in evidenza le operazioni descritte in precedenza:

```
#Generazione della sequenza di Sobol
SobolSet=qmc.Sobol(self.dim_observations,seed=self.seed)
#Calcolo del fattore di Cholesky
L=np.linalg.cholesky(self.cov)

...

# se fattore di ramificazione e' 1, nel nodo si osserva il
# valore atteso
if(self.branching_factors[i]==1):
    sample=mean[0,:]
# altrimenti, campionamento a partire da sequenza di Sobol
else:
    X=SobolSet.random(1)
    Z=[]
    for k in range(self.dim_observations):
        Z.append(ndtri(X[0][i]))
    sample=self.mean[0,:]+np.dot(L,Z)
    for k in range(self.dim_observations):
        if sample[k]<0:
            sample[k]=0
```

Come nella classe per alberi di scenari standard, i nodi vengono aggiunti con tre cicli for, ma già in precedenza si genera la sequenza di Sobol (con la funzione qmc.Sobol() del modulo scipy.stats) ed il fattore di Cholesky (con la funzione cholesky del modulo linalg di np). É da notare il fatto che questa funzione restituisce il fattore triangolare inferiore della decomposizione ($L = U^T$), dunque quando si va ad applicare la formula 2.9 non è necessario trasporre la matrice.
Le linee successive di codice mostrano come si generano i campioni nei nodi: in quelli figli di una singola ramificazione si prende il valore atteso. Altrimenti, si estrae un punto dalla sequenza di Sobol (con il metodo .random(1) applicato alla sequenza in precedenza generata), si applica componente per componente la funzione quantile della normale standard (funzione ndtri) e si applica la formula 2.9 per generare il campione (sample). Come da ipotesi iniziale, quando una componente generata è negativa, la si sostituisce col valore 0.

2.6 Risultati out of sample con albero di scenari Sobol

Dopo aver definito questa seconda classe di albero di scenari, si ripetono le simulazioni rolling horizon sugli stessi percorsi di domanda, per andare a confrontare i risultati e vedere se la tecnica di riduzione della varianza porti vantaggi o meno. Il solver utilizzato è sempre quello che sfrutta la mateuristica Fix & Relax, questa volta l'albero che gli viene passato in ingresso viene creato con la classe ScenarioTree_Sobol. Come gap di arresto si mantiene lo 0.01% ed anche in questo caso non si impone alcun limite di tempo al solver. Per quanto riguarda gli altri parametri del problema, le ipotesi che vengono fatte sono le stesse enunciate in 2.4. Il vettore dei valori attesi e la matrice di varianza covarianza vengono mantenuti costanti (come prima), gli altri parametri vengono estratti 10 volte in modo diverso (con gli stessi seed del caso precedente ovviamente), per confrontare i risultati sulle stesse configurazioni.
In analogia a quanto fatto in precedenza, il primo grafico che si riporta è quello riguardante la media mobile del costo, definita dalla formula 2.8. Come ci si aspetta, i risultati che vediamo nel grafico 2.19 sono molto simili al caso precedente. Dopo una fase di transitorio iniziale, lunga circa dieci istanti di tempo, la media del costo si assesta attorno ad un 'valore limite' attorno a cui rimane fino alla fine dell'orizzonte temporale. Questo succede in tutti e 5 i grafici della figura, dunque per tutti e 5 i fattori di ramificazione presi in esame. Si ricorda che, come nel caso precedente, ogni punto del grafico è la media di 50 medie mobili a quell'istante temporale, una per ogni coppia di seed dei parametri (10 valori) e percorso di domanda out of sample (5 valori).

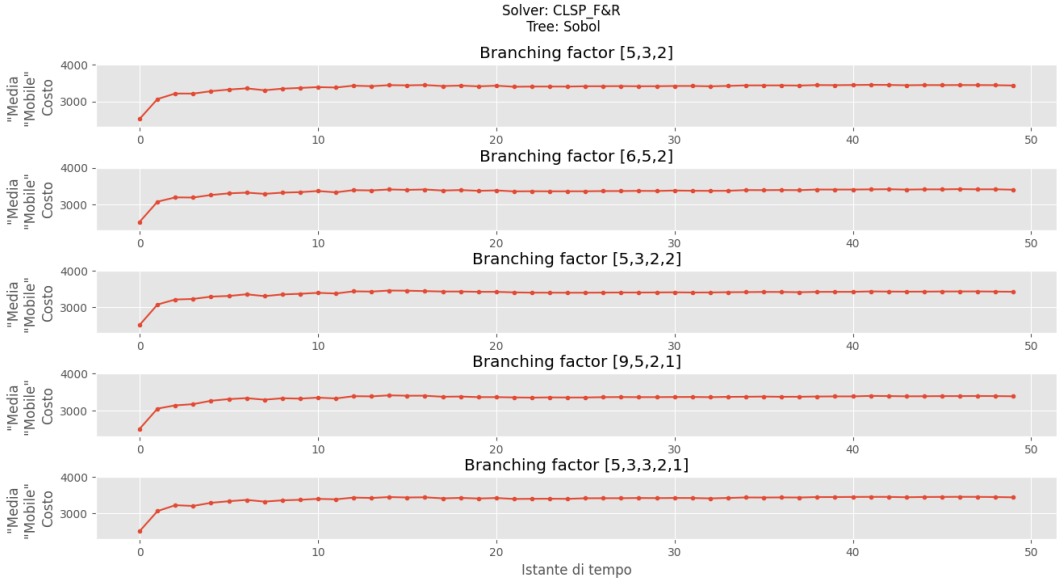

Figura 2.19. Media mobile del costo (albero Sobol)

La successiva figura 2.20, assieme alla tabella 2.4, mettono a confronto i risultati delle simulazioni in cui si è utilizzato l'albero standard e di quelle in cui si è utilizzato l'albero di scenari con sequenze a bassa discrepanza di Sobol. Ogni punto dei grafici è la media di 50 costi o tempi totali, uno per ogni coppia di seed e di percorso di domanda.
Si può vedere che i risultati sono abbastanza simili, ma l'utilizzo delle sequenze di Sobol non ha prodotto un miglioramento dei risultati. Per quanto riguarda i costi, il fattore [9,5,2,1] è quello che li minimizza tra le simulazioni con albero di Sobol, ma anche in questo caso fa peggio, seppur di poco, delle corrispondenti simulazioni con albero di scenari standard. Per tutti gli altri fattori di ramificazione, l'albero di scenari standard produce costi abbastanza inferiori rispetto a quello di tipo Sobol.
L'andamento dei tempi computazionali è abbastanza simile per i primi tre fattori, per gli ultimi due ([9,5,2,1] e [5,3,3,2,1]) le simulazioni con albero di scenari di tipo Sobol impiegano tempi sensibilmente maggiori di quelle con alberi standard. Il tempo massimo si raggiunge in corrispondenza del fattore [9,5,2,1].

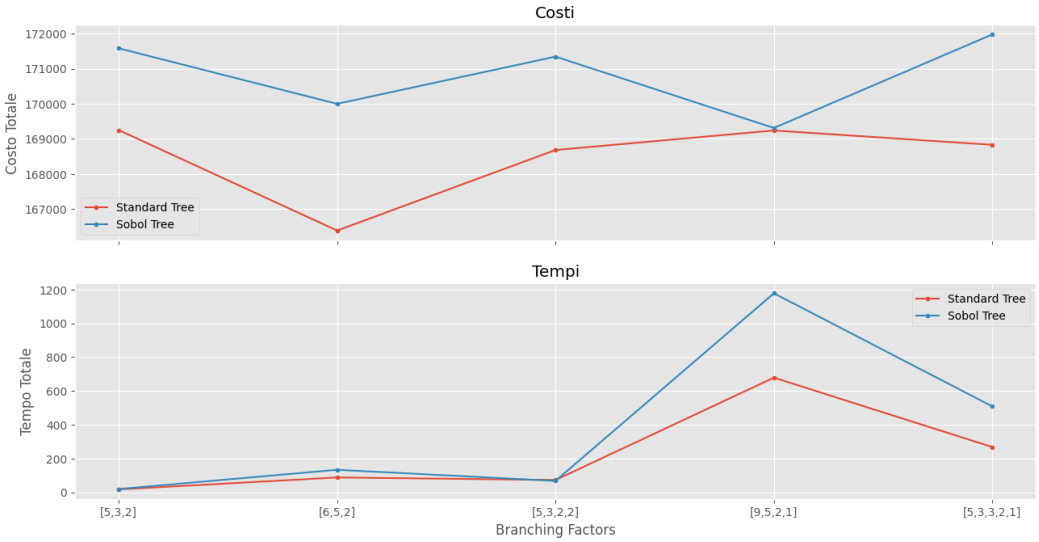

Figura 2.20. Costi e tempi totali (albero standard e Sobol)

Infine, l'ultima figura (2.21) mostra come il costo totale è distribuito tra le tre cause (magazzino, vendite perse e setup) e confronta i tre costi separati con i loro omologhi nel caso delle simulazioni con albero di scenari standard. Dal grafico in alto a sinistra si può vedere che la classifica dei singoli costi rimane la stessa del caso precedente (2.18): il costo maggiore è quello per i setup, poi quello per il magazzino, per ultimo quello dovuto alle vendite perse. Ciò che si vede cambiare, anche dagli altri tre grafici di questa figura, è la differenza tra questi costi. I costi di magazzino nelle simulazioni con albero di scenari di tipo Sobol sono molto maggiori rispetto al caso precedente, come si vede dal grafico in alto a destra. I due grafici in basso invece mostrano che sia i costi per le vendite perse

Fattore	*Costo std*	*Costo Sobol*	*Tempo std*	*Tempo Sobol*
[5,3,2]	169248.90210629985	171581.70952437609	17.05462	19.23009
[6,5,2]	166383.37822730807	169997.16943860875	87.67774	133.08096
[5,3,2,2]	168679.9936461453	171343.7798896703	72.77375	67.33468
[9,5,2,1]	169237.4710563179	169308.83153948176	678.35684	1177.46189
[5,3,3,2,1]	168828.4203705439	171976.40028392826	267.72785	508.10629

Tabella 2.4. Valori numerici della figura 2.20

che quelli per i setup sono inferiori rispetto al caso dell'albero di scenari standard. Questi risultati si possono anche verificare dalla tabella 2.5, confrontandola con quella precedente (2.2).

Figura 2.21. Costi disaggregati (albero di Sobol)

Fattore	*Costo totale*	*Costo magazzino*	*Costo vendite perse*	*Costo setup*
[5,3,2]	171581.70952437609	68248.15978760883	17754.509736767224	85579.04
[6,5,2]	169997.16943860875	67735.28616662408	17054.843271984653	85207.04
[5,3,2,2]	171343.7798896703	68033.75327558887	18493.546614081446	84816.48
[9,5,2,1]	169308.83153948176	63905.105654167404	19767.72588531434	85636.0
[5,3,3,2,1]	171976.40028392826	68580.1218297013	18144.118454226995	85252.16

Tabella 2.5. Valori costi disaggregati albero Sobol

2.7 Problema CLSP deterministico con scorte di sicurezza e solver deterministico

In tutte le sezioni precedenti si è rappresentata l'incertezza della domanda attraverso un albero di scenari, in cui in ogni nodo si campiona (nei due differenti modi visti) da una distribuzione normale multivariata. Ciò che ci si chiede è se questa rappresentazione dia effettivamente vantaggi, in termini di qualità delle decisioni prese, a fronte del fatto che rende il modello molto più complicato.
In questa sezione si prova a semplificare il modello, supponendo di non avere più un albero di scenari complesso con delle ramificazioni, bensì di averne uno in cui i fattori di ramificazione sono tutti uguali ad 1. In sostanza, ad ogni istante temporale l'albero ha un solo nodo. Questo permette di allungare notevolmente l'orizzonte temporale del problema.
La domanda osservata ad ogni nodo non si estrae più da una distribuzione normale multivariata, ma si suppone che sia sempre il suo valore atteso. Il modello è dunque ora totalmente deterministico. Per tenere conto comunque dell'incertezza della domanda, si è sperimentata l'idea di imporre delle scorte di sicurezza sul livello di magazzino per ogni prodotto, in seguito si darà una loro spiegazione più precisa.
Diamo ora una definizione precisa del modello deterministico che si vuole sperimentare:

$$min \quad \sum_i \sum_{t=0}^{T-1} \Big(h_i I_{i,t} + f_i s_{i,t} \Big) \tag{2.10}$$

$$s.t. \quad I_{i,t} = I_{i,t-1} + y_{i,t} - D_{i,t} \qquad \forall i,t \tag{2.11}$$

$$y_{i,t} \leq \Big(\sum_{k=t}^{T} D_{i,k} \Big) s_{i,t} \qquad \forall i,t \tag{2.12}$$

$$\sum_i r_i y_{i,t} + \sum_i r_i^{'} s_{i,t} \leq R \qquad \forall t \tag{2.13}$$

$$y_{i,t} \geq 0 \qquad \forall i,t \tag{2.14}$$

$$s_{i,t} \in \{0,1\} \qquad \forall i,t \tag{2.15}$$

$$I_{i,t} \geq SS_i(\alpha) \qquad \forall i,t \tag{2.16}$$

Nel seguente elenco si riportano i significati delle variabili e dei parametri del modello:

- $y_{i,t}$ è la quantità del prodotto i, prodotta all'istante temporale t
- $I_{i,t}$ è la quantità di prodotto i che resta in magazzino alla fine dell'istante temporale t (eventualmente 0) e che viene passata all'istante successivo
- $s_{i,t}$ è la variabile di setup per il prodotto i all'istante temporale t
- f_i è il costo di setup per il prodotto i
- h_i è il costo di magazzino per il prodotto i
- $D_{i,t}$ è la domanda del prodotto i all'istante temporale t; dalle ipotesi fatte, essa è sempre il valore atteso della domanda per il prodotto i
- r_i è il tempo di lavorazione del singolo pezzo del prodotto i
- $r_i^{'}$ è il tempo di setup per il prodotto i
- R è la capacità massima in termini di tempo
- $SS_i(\alpha)$ è la scorta di sicurezza che si impone al livello di magazzino del prodotto i; dipende dal parametro α
- α è il livello di sicurezza della scorta (probabilità maggiore di 0.5)
- T è la lunghezza dell'orizzonte temporale

Notiamo rispetto al caso precedente l'assenza della variabile z, quella per le domande non soddisfatte: essendo il modello deterministico, a patto che esso sia feasible, la domanda viene sempre soddisfatta. La funzione obiettivo (2.10) è la somma su tutto l'orizzonte temporale dei costi di magazzino e di setup di ogni prodotto e va ovviamente minimizzata. I primi tre vincoli (2.11,2.12,2.13) sono l'analogo nel caso deterministico dei vincoli del modello stocastico, rispettivamente sul bilanciamento del flusso, sulla quantità massima producibile e sulla disponibilità massima di tempo. I vincoli sulle variabili sono come nel caso stocastico, ad eccezione di quello sulle variabili di magazzino I (2.16). Esse sono vincolate ad essere maggiori di una scorta di sicurezza $SS(\alpha)$.
L'ultima ipotesi da fare riguarda la scelta dei valori delle scorte di sicurezza. Essa dipende dal parametro α, compreso in (0.5,1). Questo parametro indica in qualche modo il livello di sicurezza delle scorte.
Un modello in cui le scorte di sicurezza sono pari a 0 per tutti i prodotti ha livello di sicurezza 0.5 perché, avendo scelto come domanda deterministica il valore atteso, la probabilità di riuscire a soddisfare una domanda out of sample (dunque che la domanda out of sample sia minore del valore atteso) è 0.5 per tutti i prodotti. Se si vuole essere più coperti si deve imporre al livello di magazzino di ogni prodotto di essere maggiore di una scorta di sicurezza.
Per un livello di sicurezza $\alpha > 0.5$, la scorta di sicurezza per il prodotto i deve essere la differenza tra il quantile di ordine α della normale univariata ed il suo valore atteso, come mostrano le seguenti formule:

$$D_i \sim N(\mu_i, \sigma_i)$$

$$SS_i(\alpha) = q_\alpha(D_i) - \mu_i$$

In questo modo, se la domanda out of sample è maggiore del valore atteso, a magazzino è presente una scorta di sicurezza per poterla soddisfare. Il livello di sicurezza delle scorte è α, nel senso che la domanda di un prodotto può essere certamente accontentata, a patto che essa sia minore del quantile di ordine α della relativa normale univariata.
A partire dal modello descritto in precedenza e da queste ipotesi sulle scorte di sicurezza, si è definita una classe Python CLSP_Det, per definire e risolvere il problema con la formulazione di tipo deterministico.
Questa classe ha quattro metodi. Esattamente come nel caso del solver standard, nel metodo solve vengono chiamati i due metodi in comune con il caso precedente, populate e get_solution. Il seguente listato di codice mostra il metodo solve di questa classe:

```
def solve(
    self, instance, I0, time_limit=None, gap=None, verbose=False
):

    model = self.populate(instance, I0)
    return self.get_solution(instance,model,I0,
    time_limit=time_limit, gap=gap, verbose=verbose)
```

Il metodo populate definisce il modello, al suo interno sono calcolate le scorte di sicurezza e viene imposto (oltre a tutti gli altri) il vincolo sul livello di magazzino. Il seguente frammento di codice mostra questi due passaggi (all'interno del metodo populate), visto che sono quelli che caratterizzano principalmente questo tipo di formulazione:

```
SS=[]
for i in range(num_items):
    quant=norm.ppf(alpha, loc=mean[i], scale=sd[i])
    SS.append(quant-mean[i])

...

#SCORTE DI SICUREZZA MAGAZZINO
model.addConstrs(I[i,t]>=SS[i] for i in range(num_items)
for t in range(time_periods))
```

Ritornando al metodo solve, dopo aver popolato il modello chiama il metodo get_solution che lo risolve e restituisce le decisioni al primo stadio, il valore della funzione obiettivo, il tempo computazionale ed il gap raggiunto alla soluzione trovata.
Il metodo in più rispetto ai solver precedenti è stato aggiunto per cercare di evitare per quanto possibile di incorrere in modelli infeasible. Infatti, l'imposizione delle scorte di sicurezza al magazzino rende più difficile soddisfare il vincolo sulla disponibilità di tempo.
Il metodo populate2 viene chiamato all'interno di quello get_solution, il seguente listato di codice mostra quando:

```
#risoluzione del modello e tempo computazionale
```

```
start = time.time()
model.optimize()
end = time.time()
comp_time = end - start

num_items = instance.num_items

#NEL CASO IN CUI LE SCORTE DI SICUREZZA RENDANO IL PROBLEMA INFEASIBLE
#SI RIDEFINISCE IL MODELLO SENZA DI ESSE (LIVELLO DI MAGAZZINO I>=0)

if model.status == grb.GRB.INFEASIBLE:
    model.reset()
    model = self.populate2(instance, I0)


    start = time.time()
    model.optimize()
    end = time.time()
    comp_time = end - start
```

Il modello che si risolve nelle prime righe di questo listato è quello che è stato appena definito con il metodo populate, dunque con le scorte di sicurezza. Qualora questo modello non fosse feasible (condizione dell'istruzione if), allora esso viene ridefinito con il metodo populate2, eliminandole, e poi nuovamente risolto. L'idea è dunque quella di imporre le scorte di sicurezza ogni volta che è possibile, ma nel caso in cui non lo sia accontentarsi di un livello di sicurezza pari a 0.5 (come da definizione precedente).

2.8 Risultati out of sample del solver deterministico

Dopo aver definito il solver deterministico, si passa alla sua sperimentazione su scenari di domanda out of sample. Il tipo di simulazione è sempre quello di tipo rolling horizon, dunque si osserva il livello di magazzino, si prendono le decisioni di produzione usando il solver, si osserva la domanda, se possibile la si soddisfa e si calcolano i costi (costi di setup, di magazzino e di domanda non soddisfatta). É da notare che, nel caso deterministico, la domanda in sample è sempre soddisfatta (a meno di modello unfeasible), ma quando si fanno simulazioni con domande out of sample è possibile che le quantità prodotte e le scorte di sicurezza non siano sufficienti a farlo, dunque che siano presenti domande non soddisfatte $z_i > 0$ e conseguenti costi. Come valore di gap di arresto si è mantenuto lo 0.01%, non è stato imposto alcun limite di tempo al solver.
I 5 percorsi di domanda sono rimasti gli stessi, come anche le ipotesi sui parametri del problema (2.4). Questi ultimi, a differenza del vettore dei valori attesi e della matrice di varianza e covarianza (sempre gli stessi), sono stati estratti 10 volte con gli stessi seed dei casi precedenti.
Sono stati sperimentati 5 valori differenti di lunghezza dell'orizzonte temporale T ($T = 10, 20, 30, 40, 50$) e 9 valori differenti per il livello di sicurezza α (i valori compresi tra 0.55

e 0.95, con passo 0.05).
La prima figura (2.22) che si propone mette a confronto i risultati di costo totale lungo i percorsi di domanda, in corrispondenza delle differenti possibilità di coppie (T, α). Ogni singolo valore dei grafici è la media di 50 risultati ottenuti per ogni possibile coppia di seed dei parametri (10 semi) e percorso di domanda (5 percorsi). Da tutti e 4 i grafici è possibile notare che i risultati migliori in termini di costo (cioè quelli minimi) si ottengono per gli α tra 0.6 e 0.8, mentre per α superiori il costo totale cresce nettamente, ed anche per $\alpha = 0.55$ è un po' maggiore. Il livello di sicurezza imposto non deve essere troppo grande.

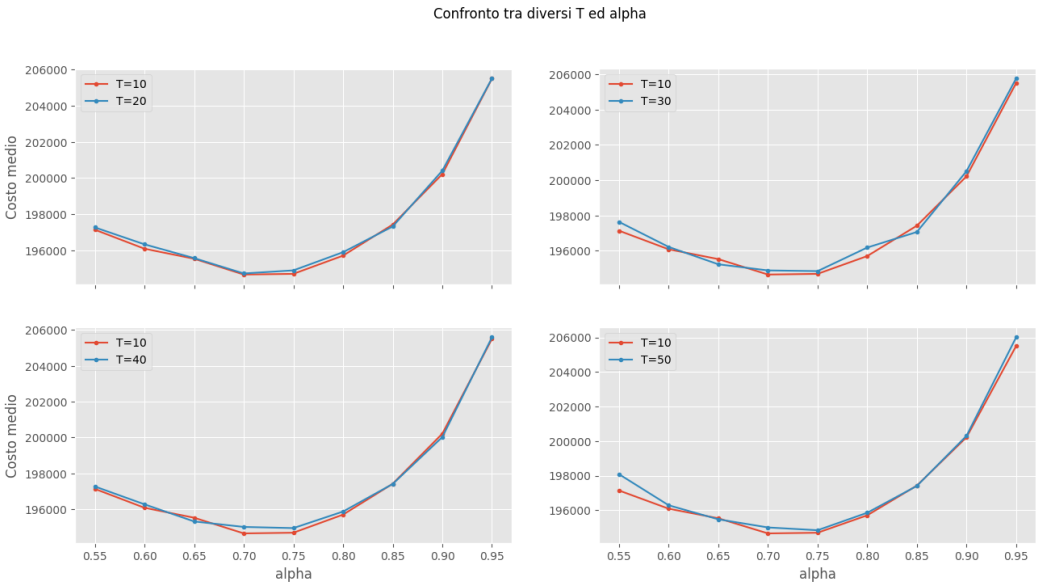

Figura 2.22. Confronto risultati a seconda di T e α

Per quanto riguarda il confronto tra le lunghezze degli orizzonti temporali, si è deciso di confrontare i risultati a coppie di T, per avere grafici senza troppe linee più facilmente comprensibili. La linea rossa rappresenta sempre i risultati ottenuti per $T = 10$ al variare degli α, mentre la linea blu rappresenta nei grafici i risultati degli altri 4 valori di T. In tutti e 4 i grafici le linee sono molto vicine e spesso si incrociano (dunque per alcuni α è meglio $T = 10$, per altri α è meglio l'altro valore di T che si mette a confronto). Sembra non esserci quindi grande differenza tra i valori di T, e soprattutto sembra non esserci grande vantaggio ad aumentare la lunghezza dell'orizzonte temporale (che si traduce in tempi maggiori per risolvere il modello). Notiamo che i risultati migliori, come minimizzazione dei costi totali, si ottengono per le coppie di (T, α) uguali a (10,0.7) e (10,0.75), con valori molto simili tra di loro.
La figura successiva mette a confronto i risultati ottenuti in precedenza dalle simulazioni del modello stocastico (con i due diversi tipi di albero di scenari) con le due migliori del

modello deterministico, appena individuate. Dalla figura 2.23 si vede immediatamente che la differenza dei risultati ottenuti è molto grande, in favore dei solver stocastici. L'utilizzo di alberi di scenari per rappresentare l'incertezza complica notevolmente il modello e dunque aumenta il tempo computazionale (come si vede dalla figura 2.24), ma si prendono decisioni di produzione migliori che riducono notevolmente i costi. Si noti che le linee che rappresentano i costi ottenuti col solver deterministico sono rette parallele all'asse delle ascisse, su cui sono rappresentati i fattori di ramificazione degli alberi utilizzati per i solver stocastici. Il solver deterministico non dipende da alcun albero di scenari, dunque il risultato ottenuto è costante.

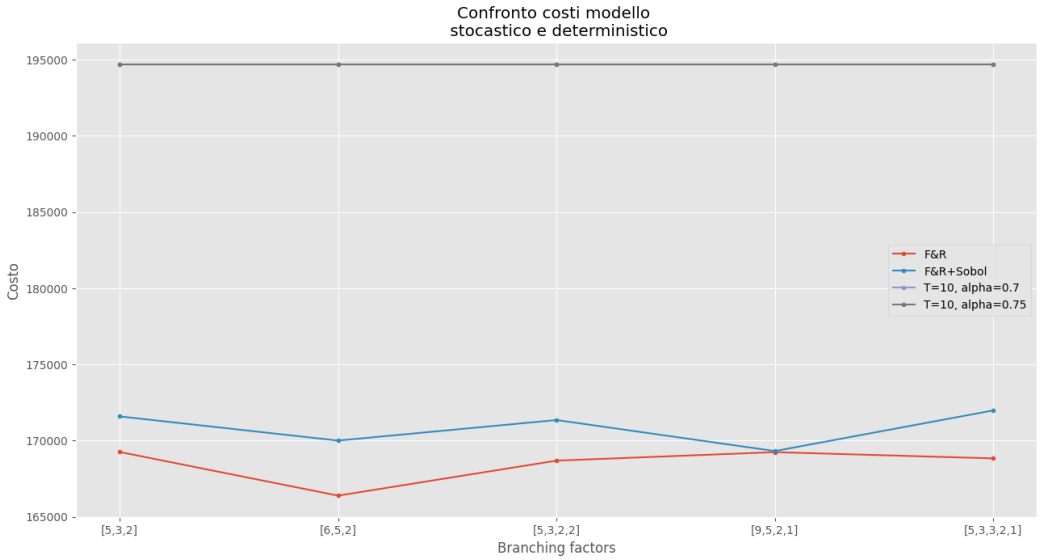

Figura 2.23. Confronto solver deterministico e stocastici

Per quanto riguarda i tempi, si vede che il tempo medio (rispetto a tutti gli α e tutti i T provati, quindi anche per orizzonti temporali abbastanza lunghi) per completare la simulazione rolling horizon è di appena 5 secondi e mezzo, nettamente inferiore a qualsiasi simulazione rolling horizon col solver stocastico, indipendentemente dal fattore di ramificazione scelto.

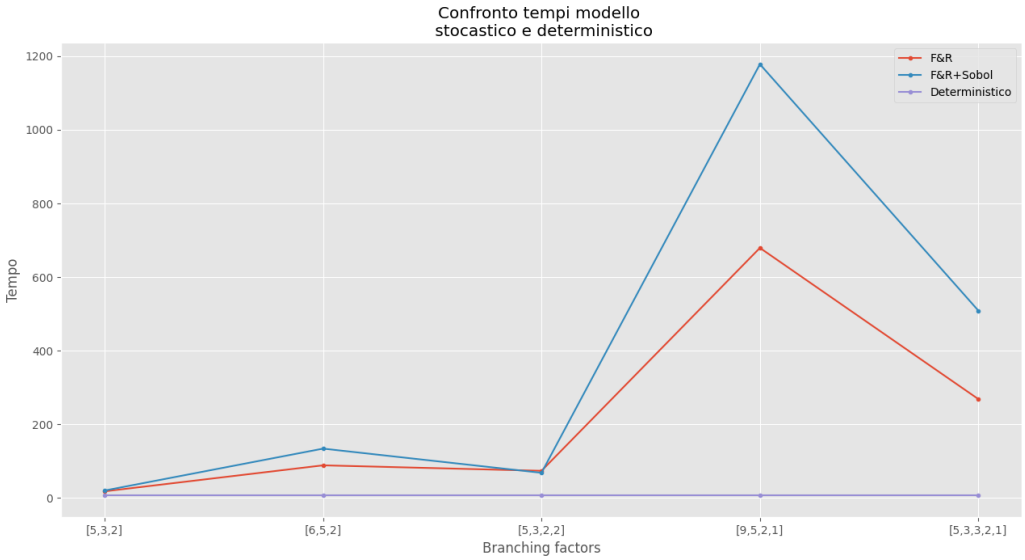

Figura 2.24. Confronto tempi dei solver stocastici e deterministico

Infine, ci si domanda quale sia la causa di questo grande peggioramento dei costi ottenuto con il solver deterministico. Dalla figura 2.25 appare evidente quale sia la ragione. Nel grafico si sono disegnati i costi disaggregati in base alle tre diverse cause (magazzino, vendite perse e setup), sia per il solver stocastico (per semplicità si è rappresentato solo il caso con albero standard, visto che l'albero di Sobol otteneva risultati simili) che per quello deterministico (con la configurazione $(T, \alpha) = (10,0.7)$).
Per quanto riguarda i costi di magazzino, il solver deterministico fa di poco peggio del solver stocastico, indipendentemente dal fattore di ramificazione. Nel caso dei costi per le vendite perse la situazione invece si ribalta: il solver deterministico ottiene risultati migliori di tutti i fattori di ramificazione usati col solver stocastico. La grande differenza tuttavia è nei costi di setup: il solver deterministico fa decisamente peggio di quello stocastico con albero standard, e da qui nasce la grande differenza di costi totali medi riscontrata nella figura 2.23.
Il solver deterministico deve mantenere il livello di magazzino di ogni prodotto sopra una soglia di sicurezza imposta, ad ogni istante di tempo. Dunque, anche se il livello di magazzino fosse di poco inferiore ad essa, la soluzione del modello imporrebbe di fare un setup, magari altrimenti evitabile, per soddisfare il vincolo delle scorte. Dato che i costi di setup sono i maggiori tra i tre tipi e soprattutto dato che sono indipendenti da quanto si produce, ma si scontano interamente anche in caso di produzione minima, appare ragionevole individuare in essi la causa della differenza.

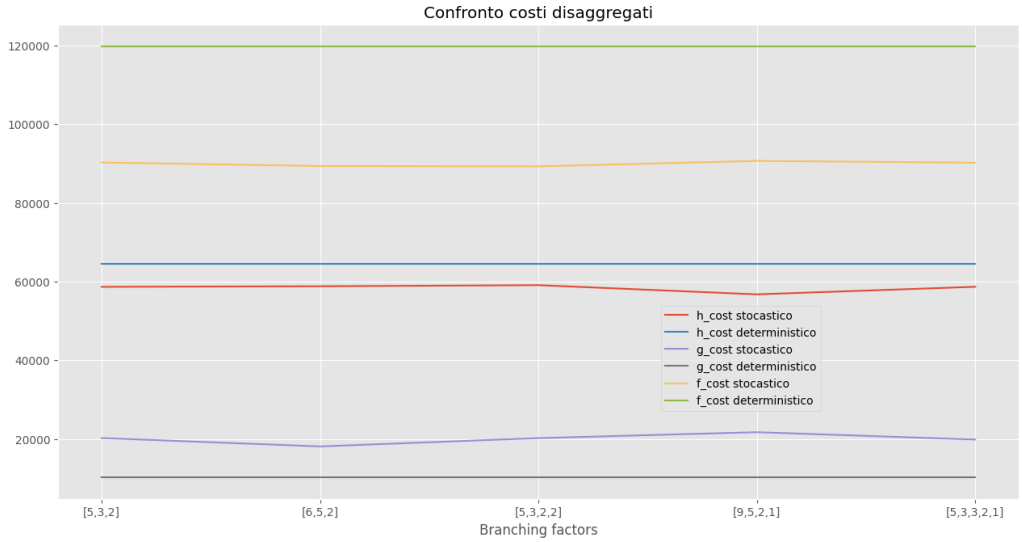

Figura 2.25. Confronto costi disaggregati dei solver stocastico e deterministico

Si è infine fatto un ultimo tentativo di modifica del modello deterministico per provare a ridurre i costi di setup. L'idea alla base è quella di imporre la scorta di sicurezza a partire dal terzo istante di tempo, lasciando nei primi due soltanto il classico vincolo di non negatività del magazzino. Visto che il solver restituisce solamente le decisioni al primo stadio (quantità da produrre e relative variabili di setup), ovvero quelle del primo istante temporale, si prova a non imporre la scorta di sicurezza in quel periodo (ed anche in quello successivo, per non avere il vincolo da soddisfare neanche subito dopo) per cercare di ridurre al minimo i setup non necessari.
Formalmente, ciò che si fa è andare a separare il vincolo 2.16 nei due seguenti vincoli:

$$I_{i,t} \geq 0 \qquad \forall i, \quad t = 0{,}1$$
$$I_{i,t} \geq SS_i(\alpha) \qquad \forall i, \quad t \geq 2$$

La variazione del solver CLSP_Det è quindi minima (solo questa nuova formulazione del vincolo delle scorte di sicurezza), dunque non si riportano altri frammenti di codice.
Si sono quindi ripetute le simulazioni rolling horizon, sempre sulle stesse configurazioni di domanda, sempre su 5 diversi valori di lunghezza dell'orizzonte temporale (10,20,30,40,50), limitandosi stavolta a 5 dei valori di α esaminati in precedenza (0.6,0.65,0.7,0.75,0.8), visto che per il primo e per i successivi i risultati peggioravano.
Il confronto tra i due solver deterministici va però nella direzione opposta rispetto a quella sperata. Il solver che non impone scorte di sicurezza nei primi due istanti di tempo ottiene risultati peggiori di quello deterministico standard, come si vede dalla figura 2.26.

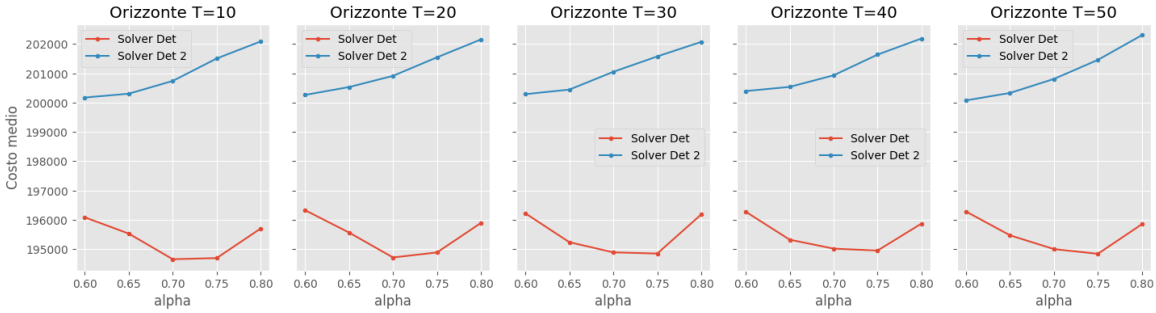

Figura 2.26. Risultati solver deterministico modificato

Per provare a capire i motivi di questi risultati, si sono rappresentati i costi ottenuti dai due solver deterministici, separandoli rispetto alla causa (costi di magazzino, di vendite perse e di setup).

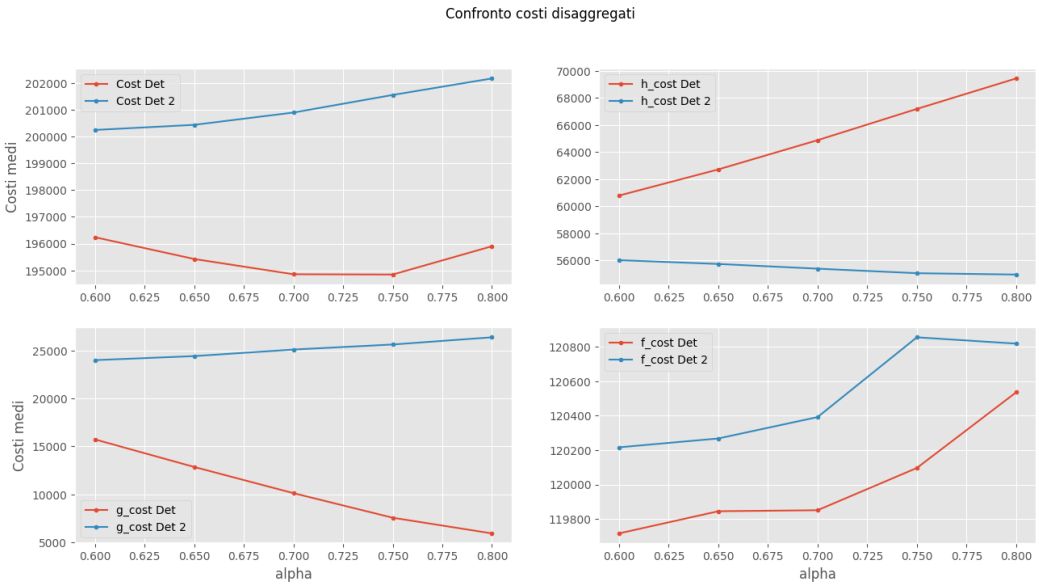

Figura 2.27. Costi disaggregati del solver deterministico modificato

Nel grafico in alto a sinistra della figura 2.27 si ripropongono i risultati di costo totale dei due solver deterministici (raggruppando le osservazioni solamente rispetto al valore

di α, che è sul'asse delle ascisse, e poi facendo la media dei costi, indipendentemente dal valore di T). Dal grafico in basso a sinistra e da quello in basso a destra si vede che il solver deterministico standard fa meglio di quello modificato per quanto riguarda i costi di vendite perse e di setup. Dal grafico in alto a destra si vede invece che per quanto riguarda i costi di magazzino, quello modificato fa meglio. Sembra dunque che si vada nella direzione opposta a quella desiderata, visto che l'obiettivo era quello di diminuire i costi di setup.
Infine, si riportano tre grafici che rappresentano i costi medi (divisi tra magazzino, setup e vendite perse) istante per istante delle simulazioni rolling horizon con i due solver deterministici, per ogni prodotto.

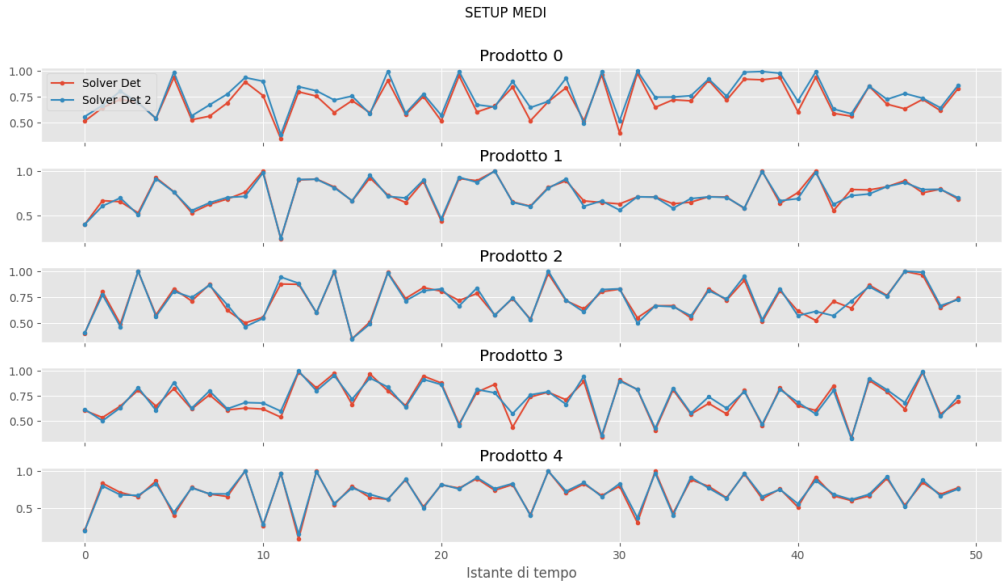

Figura 2.28. Costi di setup medi, istante per istante, divisi per prodotto

Dalla figura 2.28 si vede come il solver modificato non sia riuscito a diminuire i setup. In pochi istanti il numero medio di setup è inferiore, viceversa molto spesso è simile al caso del solver deterministico standard o superiore.
Dalle figure 2.29 e 2.30 si vede rispettivamente che il livello di magazzino medio è quasi sempre inferiore nel caso del solver deterministico modificato, mentre le vendite perse medie sono quasi sempre inferiori nel caso del solver deterministico standard.

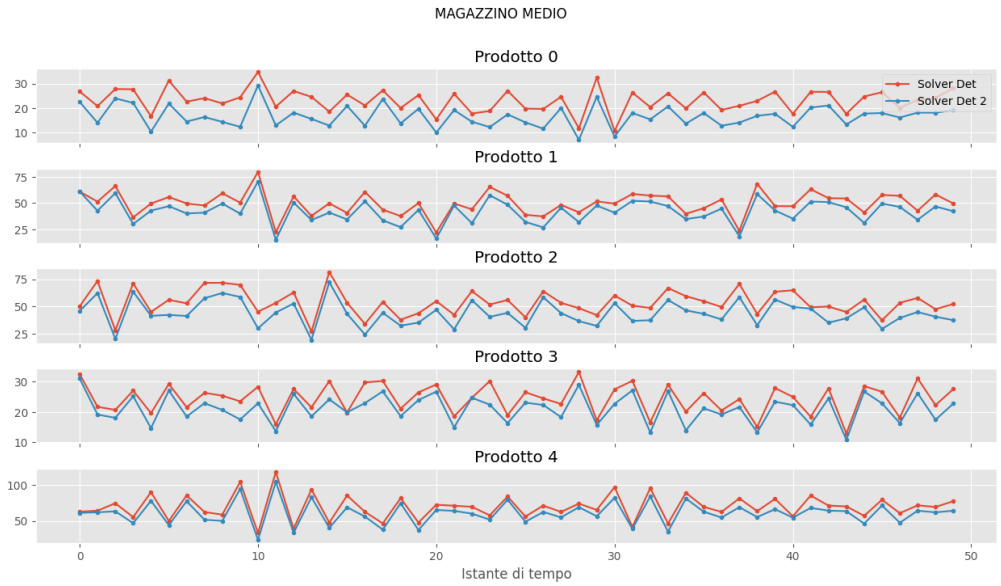

Figura 2.29. Costi di magazzino medi, istante per istante, divisi per prodotto

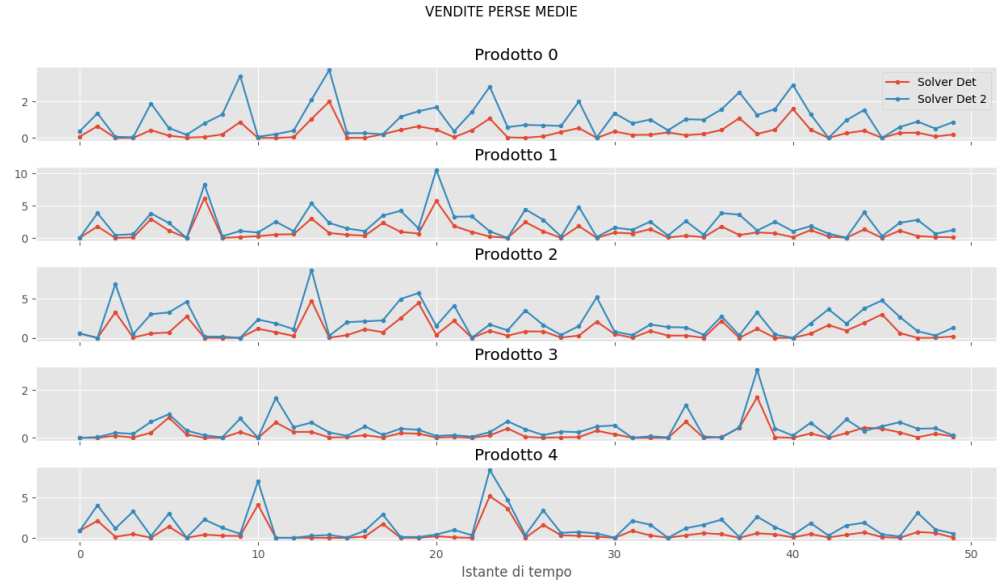

Figura 2.30. Costi di vendite perse medi, istante per istante, divisi per prodotto

Capitolo 3

Conclusioni

3.1 Confronto tra il solver standard e quello con la mateuristica Fix & Relax

Le prime sezioni del corpo del rapporto sono state dedicate alla definizione dei due solver, quello per risolvere il problema di lot-sizing soggetto a vincoli di capacità con il solver standard (che utilizza l'algoritmo branch and bound, essendo di tipo misto intero) e quello che mette in pratica la mateuristica Fix & Relax. I risultati in sample delle simulazioni effettuate hanno dato risultati positivi riguardo alla possibilità di usare il secondo solver. Esso è ovviamente molto più veloce del solver standard e, dovendolo utilizzare molte volte in una struttura di simulazioni di tipo rolling horizon, è un vantaggio importante. Inoltre, sui tipi di fattore di ramificazione presi in esame, esso si è comportato molto bene anche per quanto riguarda l'approssimazione del valore della funzione obiettivo, con differenza relativa tra i valori trovati dai due solver in media inferiore all'1%. Visto che i fattori di ramificazione erano stati scelti in modo che risolvessero in maniera accurata il problema (ovvero raggiungendo un gap nell'algoritmo branch and bound sufficientemente piccolo) col solver standard, la soluzione approssimata è sufficientemente buona da giustificare l'utilizzo del solver Fix & Relax.
Infine, avendo provato una serie di differenti configurazioni di alcuni fattori relativi ai parametri del problema, si è fatta una breve analisi statistica per capire quali tra questi fosse più influente sul tempo di soluzione del problema e sulla bontà della soluzione, per quanto riguarda il solver standard. Come ci si attendeva, il fattore più influente è il tempo medio tra ordini (T_{bo}). Esso infatti rappresenta il numero di istanti di tempo che passano tra due ordini e, se aumenta troppo, può diventare lungo come o più della profondità dell'albero di scenari utilizzato, causando problemi. Dall'analisi statistica si è visto che al suo aumentare aumenta sia il tempo necessario per la soluzione del problema che il gap medio a cui si arresta il solver standard (che spesso non raggiunge il gap limite dello 0.01%, ma si arresta per aver superato il limite di tempo).

3.2 Confronto tra albero standard ed albero di tipo Sobol

Nella parte centrale del rapporto si è utilizzato il solver standard sui percorsi di domanda out of sample, nelle simulazioni di tipo rolling horizon. Si è inizialmente utilizzato un albero di scenari di tipo standard, ovvero che in ogni nodo utilizza il metodo di campionamento standard da normale multivariata di Python. Da questo primo tipo di analisi si è visto che in questo problema aumentare troppo la complessità e la profondità degli alberi di scenari non sembra essere vantaggioso. I risultati migliori in termini di minimizzazione dei costi si sono infatti ottenuti per il fattore di ramificazione [6,5,2], che ha solo 4 istanti di tempo (compreso l'istante 0 del nodo radice) e molti meno nodi degli ultimi due esaminati. Anche come tempo i risultati sono buoni, essendo il tempo medio per completare la simulazione rolling horizon di un percorso di domanda inferiore al minuto e mezzo. I più complicati tra gli alberi trattati, ovvero gli ultimi due (con fattori di ramificazione [9,5,2,1] e [5,3,3,2,1]), impiegano tempi decisamente maggiori e non portano miglioramento nei costi totali delle simulazioni out of sample, anzi li peggiorano.
Si è in seguito provato ad utilizzare una tecnica di riduzione della varianza per campionare i valori dei nodi dell'albero di scenari e la scelta è ricaduta sulle sequenze a bassa discrepanza di tipo Sobol. I risultati di queste simulazioni, confrontate con quelle che utilizzavano il campionamento standard, evidenziano un lieve peggioramento dei risultati. I valori dei costi totali, seppur maggiori, restano comunque abbastanza simili nei due casi, nettamente migliori di quanto trovato in seguito rappresentando l'incertezza della domanda con il suo valore atteso e delle scorte di sicurezza sul livello di magazzino. Il metodo di campionamento standard di Python dunque utilizza tecniche migliori delle sequenze di Sobol, poi opportunamente trasformate in valori di normale multivariata (con uso della funzione quantile di una normale standard e della fattorizzazione di Cholesky della matrice di varianza e covarianza).
Un'altra via da sperimentare potrebbe essere l'uso di altri metodi di riduzione della varianza, ad esempio in caso di alberi di scenari caratterizzati da fattori di ramificazione con tutti valori pari si potrebbe utilizzare il campionamento antitetico, eventualmente anche accoppiato all'uso di sequenze a bassa discrepanza.
Un'altra strada che si sarebbe potuta percorrere, valida indifferentemente dal tipo di albero utilizzato per rappresentare l'incertezza, è l'utilizzo di una funzione valore dello stato finale di magazzino. Infatti, l'albero di scenari è una semplificazione che per forza di cose si assume di lunghezza finita, ed in assenza di una funzione valore si è portati a produrre in modo da lasciare il meno possibile a magazzino al termine dell'orizzonte. Nella realtà invece l'orizzonte di pianificazione non è finito, o comunque è molto lungo, dunque avere qualcosa in eccesso a magazzino potrebbe portare a vantaggi negli istanti di tempo successivi delle simulazioni rolling horizon, magari riducendo i setup, che si sono visti essere causa dei costi maggiori.

3.3 Confronto tra solver stocastico e deterministico

Nell'ultima parte del rapporto si sono confrontati i risultati delle simulazioni rolling horizon ottenuti con il solver stocastico che rappresenta l'incertezza attraverso un albero di scenari, con i risultati ottenuti con un solver più semplice, di tipo deterministico. Il dubbio era infatti quello di complicare eccessivamente il modello, senza ottenere effettivi vantaggi. Si è quindi definito un solver con domande deterministiche, tutte pari al valore atteso della distribuzione normale multivariata, ed imponendo una scorta di sicurezza al livello di magazzino di ogni prodotto.
I risultati del confronto confortano l'utilizzo degli alberi di scenari: a fronte di un modello più complicato (e dunque con tempi computazionali maggiori), i risultati su percorsi di domanda out of sample sono nettamente migliori. Il motivo della grande differenza è nei costi di setup, responsabili della quasi totalità del gap tra i costi totali delle simulazioni.
Si è infine provato ad eliminare le scorte di sicurezza dai primi due istanti di tempo (visto che il solver restituisce le decisioni al primo stadio), per evitare setup non necessari (ad esempio perché il livello di magazzino è poco sotto la scorta di sicurezza). Neanche questa variazione migliora i risultati del solver deterministico, anzi peggiorano sia i costi di setup (evidentemente il volerne fare meno all'inizio si traduce in un maggior numero in seguito) che quelli per le domande non soddisfatte, riducendo solo quelli di magazzino (coerentemente più vuoto, infatti ci sono più vendite perse e di conseguenza più necessità di fare setup per riempirlo).

Bibliografia

Daniel A. Schult Aric A. Hagberg and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, pages 11–15, 2008.

Paolo Brandimarte. *Numerical Methods in Finance and Economics: a MATLAB-Based Introduction*. Wiley, 2006a.

Paolo Brandimarte. Multi-item capacitated lot-sizing with demand uncertainty. *International Journal of Production Research*, 44(15):2997–3022, 2006b.