Business Analytics

1 CAP 1 – Introduzione alle decisioni in condizioni di incertezza

Recap Le decisioni di business si collocano spesso in contesti caratterizzati da **incertezza**, rendendo insufficiente una semplice previsione puntuale delle variabili rilevanti. Le tre componenti della **Business Analytics** chiariscono ruoli diversi ma complementari: la descrittiva analizza ciò che è accaduto, la predittiva costruisce modelli per descrivere il futuro, mentre la prescrittiva individua **azioni ottimali** tenendo conto delle conseguenze economiche.

La qualità di una previsione dipende dalla **funzione di perdita** associata all'errore. Penalità diverse conducono a previsioni ottimali diverse: il valore atteso è appropriato solo in presenza di costi simmetrici, mentre penalità asimmetriche portano naturalmente a soluzioni basate su **quantili**. Non esiste quindi una previsione universalmente ottima, ma solo scelte coerenti con il criterio decisionale adottato.

Nei problemi decisionali l'obiettivo diventa la **scelta di un vettore di decisioni** sotto incertezza. Quando i fattori di rischio non sono probabilisticamente modellabili si adotta un approccio **robusto**, basato su scenari peggiori; quando invece è disponibile una distribuzione di probabilità si ottiene un **problema di ottimizzazione stocastica**. In generale, sostituire l'incertezza con valori medi non preserva l'ottimalità della decisione.

Il modello del **newsvendor** evidenzia come la decisione ottimale derivi dal bilanciamento tra **costi di underage e overage**, mostrando che la domanda media non è un criterio corretto. Questo esempio chiarisce il legame diretto tra struttura dei costi e decisioni sotto incertezza.

I **vincoli probabilistici** consentono di imporre il rispetto dei vincoli solo con una certa probabilità, ma possono generare insiemi ammissibili non convessi e problemi difficili da trattare. Gli **alberi decisionali** permettono invece di rappresentare decisioni **adattive**, distinguendo tra nodi decisionali e nodi casuali e risolvendo il problema tramite **backward induction** sulla base del valore atteso.

Il valore dell'informazione viene quantificato attraverso **EVPI** e **VSS**, che misurano rispettivamente il beneficio teorico dell'informazione perfetta e il guadagno ottenuto risolvendo il problema stocastico rispetto a una soluzione deterministica a valore atteso.

L'introduzione ai **modelli a due stadi** chiarisce la distinzione tra decisioni **here-and-now** e decisioni **wait-and-see**, mostrando il ruolo delle **decisioni di ricorso**. Questa struttura si estende ai modelli multistadio, nei quali le decisioni dipendono dall'intera storia delle realizzazioni aleatorie e richiedono il rispetto della **non-anticipatività**.

Nei modelli multistadio emerge il problema della **generazione degli scenari** e della crescita dell'albero, che rende necessarie tecniche di campionamento, riduzione e progettazione controllata degli scenari. La qualità di una soluzione va quindi valutata anche in termini di **stabilità in-sample e out-of-sample**, verificando che le prestazioni non dipendano in modo critico dall'albero utilizzato.

Si analizzano le **proprietà di convessità** dei modelli stocastici: i vincoli probabilistici non preservano la convessità in generale, mentre nei modelli a due stadi con ricorso lineare la funzione di ricorso risulta convessa, proprietà fondamentale per lo sviluppo di metodi di soluzione basati su decomposizione.

Introduzione Si distinguono tre principali tipologie di *Business Analytics*

- **Business Analytics descrittiva**: ha l'obiettivo di **analizzare** e sintetizzare i **dati storici** al fine di comprendere che cosa è accaduto in passato (e.g. overbooking nel trasporto aereo).
- **Business Analytics predittiva**: mira a **stimare eventi o variabili future** sulla base di modelli (e.g. distinzione tra *forecasting*–previsione non sotto il mio controllo– e *prediction*).
- **Business Analytics prescrittiva**: si concentra sul **supporto alle decisioni**, indicando quale azione dovrebbe essere intrapresa per ottimizzare una misura (e.g. *operations management*).

1.1 Motivazione: perché considerare l'incertezza

Previsioni puntuali e decisioni Un punto di partenza naturale nello studio delle decisioni in condizioni di incertezza è la **costruzione di una previsione puntuale** di una variabile aleatoria. Sia X una v.a. reale, con distribuzione di probabilità nota. Una **previsione puntuale** consiste nella scelta di un valore $x \in \mathbb{R}$, che rappresenta una **stima ex ante** della realizzazione futura di X. La **qualità di una previsione** deve essere misurata in funzione del **costo associato all'errore** di previsione, che dipende dalla discrepanza tra il valore scelto x e la realizzazione effettiva di X.

Errore quadratico medio (MSE) Una scelta naturale per misurare il **costo dell'errore di previsione è la perdita quadratica**. In questo caso, il costo associato alla previsione puntuale x è $(X-x)^2$. Il *problema decisionale* consiste quindi nel **minimizzare l'errore quadratico medio**: $\mathbb{E}\big[(X-x)^2\big]$. Sviluppando il valore atteso, si ottiene

$$MSE(x) \doteq \mathbb{E}\big[(X-x)^2\big] = \mathbb{E}[X^2] - 2x\,\mathbb{E}[X] + x^2.$$

La condizione di ottimalità del primo ordine implica che la *previsione ottima coincide con il valore atteso della variabile aleatoria*, $x^* = \mathbb{E}[X]$. Questo risultato mostra che l'uso del valore atteso come previsione puntuale è ottimale solo sotto l'ipotesi di una perdita quadratica (simmetrica).

Perdita assoluta e penalità asimmetriche Un criterio alternativo consiste nel misurare l'errore di previsione tramite la **deviazione assoluta**, $\mathbb{E}\big[|X-x|\big]$. In questo caso, la soluzione ottima è data dalla **mediana della distribuzione** di X, anziché dal valore atteso.
Nelle applicazioni economiche, errori di previsione positivi e negativi possono avere impatti diversi. È quindi naturale introdurre una **funzione di perdita asimmetrica** del tipo

$$\mathbb{E}\big[c_u(X-x)^+ + c_o(X-x)^-\big],$$

dove $(\cdot)^+ = \max\{0,\cdot\}$ e $(\cdot)^- = \max\{0,-\cdot\}$, mentre c_u e c_o rappresentano rispettivamente il **costo di sottostima e di sovrastima**. La soluzione ottima è un *quantile della distribuzione di* X, il cui livello dipende dal rapporto tra c_u e c_o.

Il **messaggio essenziale** è che una previsione puntuale, tipicamente basata sul valore atteso, non è sufficiente per prendere decisioni ottimali in condizioni di incertezza. La previsione ottimale dipende dalla funzione di perdita adottata e quindi dall'impatto economico dell'errore di previsione. In generale, *non esiste una previsione "migliore" in senso assoluto, ma solo previsioni coerenti con uno specifico criterio decisionale.*

Modelli decisionali in ambito business L'obiettivo non è la previsione di una variabile aleatoria, ma la **selezione di un vettore di decisioni** $x \in X$ che ottimizzi una funzione economica in presenza di *fattori di rischio*. Se ξ denota un vettore di variabili aleatorie che influenzano il risultato economico, il problema può essere formulato come un **problema di ottimizzazione sotto incertezza**

$$\min_{x \in S} f(x, \xi).$$

Se non si dispone di informazioni probabilistiche sui fattori di rischio e si conosce solo un insieme di incertezza U, e il set di ammissibilità S, il problema assume la forma di un **worst-case robust optimization problem**

$$\min_{x \in S} \max_{\xi \in U} f(x, \xi).$$

Se invece i fattori di rischio sono modellati come variabili casuali con distribuzione nota, si ottiene un **problema di ottimizzazione stocastica**

$$\min_{x \in S} \mathbb{E}_{\mathbb{P}}\big[f(x, \tilde{\xi})\big].$$

Questa formulazione mette in evidenza che, in generale, $\mathbb{E}_{\mathbb{P}}\big[f(x,\tilde{\xi})\big] \neq f\big(x, \mathbb{E}_{\mathbb{P}}[\tilde{\xi}]\big)$, e giustifica la necessità di *modelli decisionali che tengano esplicitamente conto dell'incertezza*. La formulazione di questo modello suggerisce un problema decisionale statico, in base al quale prendiamo una decisione x prima della realizzazione della variabile casuale $\tilde{\xi}$, ma non c'è modo di adattare la decisione dopo il verificarsi dell'evento casuale.

Differenza tra rischio e incertezza Il **rischio** è definito come la presenza di variabili aleatorie con distribuzione di probabilità nota, mentre l'**incertezza** si riferisce a situazioni in cui tali distribuzioni non sono note o non possono essere stimate con precisione.

1.2 Modelli decisionali statistici

Il modello classico del newsvendor Il **modello classico del *newsvendor*** rappresenta un problema di decisione sotto condizione di incertezza in cui *una decisione deve essere presa prima dell'osservazione di una variabile aleatoria*. Sia D una variabile casuale non negativa che rappresenta la **domanda**, con distribuzione di probabilità nota. Il decisore sceglie una **quantità** $q \geq 0$, a **prezzo** c prima di osservare la realizzazione di D. Ogni pezzo viene venduto a un **prezzo** $p > c$ durante la finestra di vendita, e ad un **prezzo ridotto** successivamente $p_u < c$.
L'intuizione potrebbe suggerire di porre semplicemente $q = \mathbb{E}\big[D\big]$ per massimizzare il valore atteso del profitto. Ma questo risulta essere sbagliato. In realtà, il risultato non sorprende se introduciamo due tipi di costo:

- Se $q < D$, avremo un **costo opportunità** $m = p - c$, cioè il margine di profitto per la parte di domanda non soddisfatta (**shortfall**).
- Se $q > D$, avremo un **costo dell'invenduto** $c_u = c - p_u$, legata alla svendita della giacenza residue (**surplus**).

Espressione generale del profitto e riscrittura con costi di underage/overage La **formulazione tramite funzione di perdita** può essere affiancata da una scrittura esplicita del profitto, utile per evidenziare la decomposizione in termini di *underage* e *overage*. L'espressione generale del profitto è

$$\pi(q,d) = -cq + p\min(q,d) + p_u \max(q-d,0) = -cq + p\min(q,d) + p_u(q-d)^+.$$

Usando l'identità $q = \min(q,d) + (q-d)^+$, il profitto può essere riscritto come $\pi(q,d) = c_u \min(q,d) - c_o(q-d)^+$, dove definiamo il costo di underage $c_u = p - c$ e il costo di overage $c_o = c - p_u$.
Se l'**incertezza** sulla **domanda** è modellata da una v.a. continua con densità $f_D(x)$, il **profitto atteso** è

$$\mathbb{E}[\pi(q,D)] = c_u \left(\int_0^q x f_D(x)\,dx + \int_q^{+\infty} q f_D(x)\,dx \right) - c_o \int_0^q (q-x) f_D(x)\,dx.$$

Teorema – Regola di Leibniz Consideriamo una funzione di due variabili $g(q,x)$ e definiamo una funzione della sola q come

$$G(q) = \int_{h_1(q)}^{h_2(q)} g(q,x)\,dx.$$

Notiamo che anche gli estremi di integrazione sono funzioni di q. Sotto opportune ipotesi di continuità, la **regola di Leibniz** permette di scrivere

$$\frac{dG}{dq}(q) = \int_{h_1(q)}^{h_2(q)} \frac{\partial g}{\partial q}(q,x)\,dx + g\big(q,h_2(q)\big)\,h_2'(q) - g\big(q,h_1(q)\big)\,h_1'(q).$$

Nel caso del **profitto atteso del newsvendor**, tramite la regola di Leibniz otteniamo

$$\frac{d\,\mathbb{E}[\pi(q,D)]}{dq} = c_u \left(q f_D(q) + \int_q^{+\infty} f_D(x)\,dx - q f_D(q) \right) - c_o \int_0^q f_D(x)\,dx$$

$$= c_u \int_q^{+\infty} f_D(x)\,dx - c_o \int_0^q f_D(x)\,dx = c_u\big(1 - F_D(q)\big) - c_o F_D(q) = 0,$$

dove $F_D(x) \doteq \mathbb{P}\{D \leq x\}$ è la *funzione di distribuzione cumulativa* della domanda. Ponendo a zero la derivata prima, ricaviamo immediatamente

$$F_D(q^*) = \frac{c_u}{c_u + c_o}.$$

Come verifica, risulta utile controllare la **derivata seconda**

$$\frac{d^2\mathbb{E}[\pi(q,D)]}{dq^2} = -c_u f_D(q) - c_o f_D(q) < 0 \qquad \forall q,$$

in quanto la funzione di densità non può assumere valori negativi. Ciò dimostra la **concavità del profitto atteso rispetto a** q.

Modelli con vincoli probabilistici (chance-constrained models) L'idea alla base dei *chance-constrained models* è che un **vincolo stocastico non debba necessariamente essere soddisfatto in ogni realizzazione dell'incertezza**, ma solo con una probabilità sufficientemente elevata. In particolare, un vincolo del tipo $g(x,\tilde{\xi}) \leq 0$ è considerato *accettabile* se risulta soddisfatto con probabilità almeno pari a un livello prefissato. Sono considerati **vincoli probabilistici individuali** o **congiunto**

$$\mathbb{P}\{g_j(x,\tilde{\xi}) \leq 0\} \geq 1 - \alpha_j,\ j \in [m], \qquad \mathbb{P}\{\mathbf{g}(\mathbf{x},\tilde{\boldsymbol{\xi}}) \leq \mathbf{0}_m\} \geq 1 - \alpha,$$

dove $\mathbf{g}$ è una **funzione vettoriale**. L'intuizione potrebbe suggerire che, *se le funzioni g_j sono convesse rispetto a x per ogni realizzazione di $\tilde{\xi}$, allora anche il vincolo probabilistico dovrebbe preservare la convessità*. Tuttavia, **questo non è vero in generale**: *l'insieme ammissibile di un problema con vincoli probabilistici può risultare non convesso*, in quanto definito come unione di insiemi ammissibili associati a diversi scenari. Per questo motivo, i modelli chance-constrained possono risultare difficili da trattare dal punto di vista computazionale e spesso vengono approssimati tramite formulazioni di tipo robusto. Nonostante ciò, essi rappresentano uno strumento naturale per modellare problemi statici in cui non è possibile adattare le decisioni dopo la realizzazione dell'incertezza.

1.3 Alberi decisionali

Dalle decisioni statiche a quelle adattive Un modo naturale per introdurre **decisioni adattive in condizioni di incertezza** è l'utilizzo degli **alberi di decisione**. Gli alberi di decisione *consentono di rappresentare in modo esplicito la sequenza temporale delle decisioni e delle realizzazioni aleatorie*, evidenziando come le decisioni possano essere adattate sulla base delle informazioni che si rendono disponibili nel tempo. Un albero di decisione è costituito da due tipi fondamentali di nodi

- **Nodi decisionali**, rappresentati da quadrati, che corrispondono a **scelte discrete tra alternative mutuamente esclusive**. In questi nodi il decisore deve selezionare una sola azione tra quelle disponibili.
- **Nodi casuali**, rappresentati da cerchi, che descrivono la **realizzazione di esiti aleatori**. A ciascun esito i è associata una probabilità π_i, e la somma delle probabilità degli esiti che partono da uno stesso nodo casuale è pari a 1.

Un albero di decisione è composto da una *sequenza di nodi decisionali e nodi casuali*, che in genere si **alternano**. L'albero termina con **nodi terminali**, rappresentati da punti, ai quali sono associati i **risultati finali**, tipicamente espressi in termini monetari.

Strategia e criterio di ottimalità Risolvere un problema rappresentato mediante un albero di decisione significa individuare una **strategia**, ovvero una *regola che specifica quale decisione assumere in ciascun nodo decisionale che può essere visitato.* Assumendo che i risultati abbiano natura monetaria, il criterio più naturale è la **massimizzazione del valore monetario atteso EMV** (*Expected Monetary Value*).

Quando un nodo decisionale è seguito da nodi casuali, ciascun nodo casuale viene etichettato con il proprio **valore monetario atteso**. Questo consente di confrontare le alternative disponibili nel nodo decisionale. La procedura di soluzione procede **a ritroso nel tempo** (*backward induction*), iniziando dai nodi terminali e risalendo progressivamente verso la radice dell'albero. *Un nodo può essere valutato solo quando tutti i suoi successori sono già stati valutati.*

Esempio: introduzione di un nuovo prodotto Un tipico esempio di applicazione degli alberi di decisione riguarda la scelta se **lanciare un nuovo prodotto**, vendere una licenza o acquisire **informazioni aggiuntive** tramite un'indagine di mercato. L'indagine fornisce informazioni che modificano le stato di conoscenza, ma non la probabilità di successo del prodotto. La soluzione ottimale si ottiene confrontando il valore monetario atteso delle diverse strategie, tenendo conto del costo dell'informazione e delle decisioni ottimali successive ai diversi esiti osservati.

1.4 EVPI and VSS

Problema statico di ottimizzazione stocastica Si consideri il valore ottimo di un problema di ottimizzazione stocastica statica:

$$f^* = \min_{x \in S} \mathbb{E}_P \left[f(x, \tilde{\xi}) \right].$$

In questo contesto la decisione viene presa **here-and-now**, prima della realizzazione dell'incertezza.

Ci si può chiedere come migliorerebbe il valore ottimo se fosse possibile **posticipare la decisione** e osservare prima lo scenario realizzato. Formalmente, ciò equivale a scambiare l'operatore di minimo con il valore atteso:

$$f^*_{PI} = \mathbb{E}_P \left[\min_{x \in S} f(x, \tilde{\xi}) \right].$$

Il pedice PI indica che questo valore corrisponde all'ottimo ottenibile in presenza di **informazione perfetta**.

Valore atteso dell'informazione perfetta (EVPI) Per un problema di minimizzazione vale la disuguaglianza $f^*_{PI} \leq f^*$, poiché l'accesso all'informazione perfetta non può peggiorare la soluzione. La differenza tra i due valori ottimi definisce il **valore atteso dell'informazione perfetta**:

$$EVPI = f^* - f^*_{PI}.$$

L'EVPI fornisce una misura quantitativa dell'**impatto dell'incertezza** sul problema decisionale. L'EVPI misura il beneficio teorico della **chiaroveggenza**. In pratica, l'accesso a informazione perfetta è estremamente raro; pertanto, l'EVPI va interpretato come un **limite superiore** al valore che l'informazione può avere.

Esempio: scelta di una strategia di investimento Un esempio illustrativo riguarda la scelta tra due strategie di investimento alternative, una *aggressiva* e una *conservativa*, in presenza di diversi stati possibili dell'economia. Nel caso **here-and-now**, la decisione viene presa prima di osservare lo stato dell'economia e la strategia viene scelta massimizzando il rendimento atteso. Nel caso **wait-and-see**, la strategia ottimale può essere selezionata dopo aver osservato lo scenario realizzato, ottenendo un valore atteso maggiore. La differenza tra i due valori rappresenta l'EVPI.

Soluzione a valore atteso Un approccio più semplice consiste nel **trascurare l'incertezza** e sostituire le variabili aleatorie con i loro valori attesi. In questo modo si ottiene il problema deterministico a valore atteso:

$$f^*_{EV} = \min_{x \in S} f\left(x, \mathbb{E}_P[\tilde{\xi}]\right),$$

la cui soluzione è detta **expected-value solution** e viene indicata con $\bar{x}$.

La soluzione $\bar{x}$ deve essere valutata nel contesto stocastico originale. Il costo risultante è la variabile aleatoria $f(\bar{x}, \tilde{\xi})$, di cui si considera il valore atteso:

$$f_{EEV} = \mathbb{E}_P\left[f(\bar{x}, \tilde{\xi})\right].$$

Il confronto corretto non è tra f^*_{EV} e f^*, ma tra f_{EEV} e f^*. Infatti, il **valore della soluzione stocastica** è definito come

$$VSS = f_{EEV} - f^*,$$

nel caso di minimizzazione (con scambio dei termini nel caso di massimizzazione). È possibile dimostrare che $VSS \geq 0$. Quando il VSS è elevato, *lo sforzo computazionale richiesto per risolvere il problema stocastico completo è giustificato dal miglioramento ottenuto rispetto alla soluzione deterministica a valore atteso.*

Collegamento con modelli multi-stadio Il concetto di VSS viene ulteriormente chiarito attraverso esempi numerici e rappresenta un ponte naturale verso i **modelli di ottimizzazione a due stadi e multi-stadio**, nei quali le decisioni possono essere adattate dopo l'osservazione dell'incertezza.

1.5 Un'introduzione a un modello a due stadi: Assemble-to-order (ATO)

ATO Si consideri un esempio didattico di **pianificazione della produzione** in cui i prodotti finiti sono ottenuti assemblando un insieme di componenti. Ogni prodotto finale è caratterizzato da un insieme di caratteristiche, per ciascuna delle quali sono disponibili opzioni alternative che richiedono componenti diversi. Dalla combinazione di un numero limitato di componenti può derivare un numero molto elevato di prodotti finali.

Per questo motivo, non è possibile proteggersi dall'incertezza della domanda mantenendo scorte di prodotti finiti. D'altra parte, una strategia puramente **make-to-order** può risultare poco efficace a causa dei lunghi tempi di approvvigionamento dei componenti. In un ambiente **assemble-To-Order**, in cui l'assemblaggio è rapido, è invece possibile mantenere a magazzino i componenti e assemblare i prodotti finali solo dopo aver osservato la domanda.

Questo contesto suggerisce naturalmente una **strategia a due stadi**:

- Pianificazione della produzione dei componenti **here-and-now**, sotto incertezza sulla domanda.
- Assemblaggio dei prodotti finali in modalità **wait-and-see**, dopo l'osservazione degli ordini.

Modello deterministico a valore atteso *Ignorando l'incertezza della domanda* e *sostituendola con il valore medio*, si introducono le seguenti variabili decisionali

- $x_i \in \mathbb{Z}_+$, $i = 1, \ldots, n_i$, numero di componenti prodotti.
- $y_j \in \mathbb{Z}_+$, $j = 1, \ldots, n_j$, numero di prodotti finali assemblati.

Il **problema di massimizzazione del profitto** può essere formulato così

$$\begin{aligned}
\max \quad & -\sum_{i\in[n_i]} C_i x_i + \sum_{j\in[n_j]} P_j y_j \\
\text{s.t.} \quad & \sum_{i\in[n_i]} T_{im} x_i \le L_m, && m \in [n_m] \\
& y_j \le \bar{d}_j, && j \in [n_j] \\
& \sum_{j\in[n_j]} G_{ij} y_j \le x_i, && i \in [n_i] \\
& x_i,\, y_j \in \mathbb{Z}_+, && i \in [n_i],\ j \in [n_j].
\end{aligned}$$

Limiti della soluzione deterministica La soluzione ottenuta è facilmente interpretabile ma estremamente **sbilanciata**. Essa rappresenta una scommessa sulla domanda del prodotto più redditizio e può risultare molto rischiosa se la domanda effettiva si discosta dal valore medio.

Modello stocastico a due stadi Per tenere conto dell'incertezza, si introducono:

- Una domanda scenario-dipendente d_j^s con probabilità π_s;
- Variabili di secondo stadio $y_j^s \in \mathbb{Z}_+$ per l'assemblaggio adattivo.

Il **modello stocastico a due stadi** è

$$\begin{aligned}
\max \quad & -\sum_{i\in[n_i]} C_i x_i + \sum_{s\in[n_s]} \pi^s \left(\sum_{j\in[n_j]} P_j y_j^s \right) \\
\text{s.t.} \quad & \sum_{i\in[n_i]} T_{im} x_i \le L_m, && m \in [n_m] \\
& y_j^s \le \bar{d}_j^s, && j \in [n_j],\ s \in [n_s] \\
& \sum_{j\in[n_j]} G_{ij} y_j^s \le x_i, && i \in [n_i],\ s \in [n_s] \\
& x_i, y_j^s \in \mathbb{Z}_+, && i \in [n_i] j \in [n_j],\ s \in [n_s].
\end{aligned}$$

La soluzione stocastica produce quantità di componenti meno estreme e decisioni di assemblaggio che variano a seconda dello scenario. Le variabili di secondo stadio rappresentano piani di contingenza e non costituiscono output operativi immediati.

Confronto corretto tra le soluzioni Il confronto diretto tra f^* e f_{EV} non è significativo. Occorre fissare le decisioni al primo stadio e risolvere, per ciascuno scenario S, il **problema al secondo stadio**

$$\begin{aligned}
R^s(\mathbf{x}^\circ) = \max \quad & \sum_{j\in[n_j]} P_j y_j^s \\
\text{s.t.} \quad & y_j^s \le d_j^s, && j \in [n_j] \\
& \sum_{j\in[n_j]} G_{ij} y_j^s \le x_i^\circ, && i \in [n_i] \\
& y_j^s \in \mathbb{Z}_+, && j \in [n_j].
\end{aligned}$$

Dove $R^s(\mathbf{x}^\circ)$ è il **ricavo ottimo** che otteniamo nello scenario s, dato il vettore di decisione di **primo stadio** $\mathbf{x}^\circ$, sfruttando in modo ottimale i componenti disponibili per soddisfare la domanda. Si noti che, in questo modello, la disponibilità dei componenti $\mathbf{x}^\circ$ è data, a seconda dei casi, dal modello **stocastico** oppure dal modello **deterministico**. In entrambi i casi, il **ricavo atteso** risultante è dato da

$$\sum_{s \in S} \pi^s R^s(\mathbf{x}^\circ).$$

1.6 Modelli a due stadi

Dall'ATO ai modelli a due stadi con ricorso Il modello *assemble-to-order* fornisce una buona introduzione ai **modelli di programmazione lineare stocastica a due stadi con ricorso** (*recourse*), che rappresentano l'esempio più semplice di modelli di ottimizzazione adattivi. In un modello a due stadi si distinguono due tipi di decisioni:

- Decisioni **here-and-now** (primo stadio) x, che devono essere prese sotto incertezza (ad esempio al tempo $t = 0$);
- Decisioni **wait-and-see** (secondo stadio) $y(\tilde{\xi})$, che possono essere prese dopo aver osservato la realizzazione dei fattori di rischio $\tilde{\xi}$ (ad esempio al tempo $t = 1$).

Le decisioni di secondo stadio sono anche dette **decisioni di ricorso**. Si osserva immediatamente che tali decisioni sono in realtà **politiche decisionali**, cioè funzioni che mappano la realizzazione dei fattori di rischio in una decisione ammissibile; pertanto appartengono a uno spazio **infinito-dimensionale**.

Formulazione annidata del modello Il modello può essere formulato come **due problemi di ottimizzazione annidati**

$$\begin{aligned} \min_x \; & c^T x + Q(x) \\ \text{s.t.} \; & Ax = b \\ & x \geq 0, \end{aligned}$$

dove la **funzione di ricorso** è definita come $Q(x) = \mathbb{E}_{\mathbb{P}}\left[Q(x, \tilde{\xi})\right]$. Il **problema al secondo stadio** è

$$\begin{aligned} Q(x, \xi) = \min_y \; & q(\xi)^T y \\ \text{s.t.} \; & Wy = h(\xi) - T(\xi)x \\ & y \geq 0. \end{aligned}$$

Ricorso fisso e non-linearità indotta dal ricorso Nel problema di secondo stadio sia la decisione di primo stadio x sia la realizzazione ξ dei fattori di rischio sono dati. Risolvendo il secondo stadio per ogni realizzazione ξ, si definisce implicitamente una funzione $y(\tilde{\xi})$, mostrando che le variabili di secondo stadio sono effettivamente funzioni. In questa formulazione la **matrice di ricorso** W non dipende da variabili aleatorie e si parla di **ricorso fisso** (*fixed recourse*). La formulazione evidenzia che **la programmazione lineare stocastica con ricorso è, in generale, un problema di programmazione non lineare** (poiché $Q(x)$ dipende da x tramite un problema di ottimizzazione interno).

La funzione $Q(x)$ è un valore atteso rispetto alla distribuzione congiunta di $\tilde{\xi}$; se le variabili aleatorie sono continue, essa è un integrale multidimensionale. Inoltre, tale integrale riguarda una funzione che non è nota in forma chiusa, poiché è definita implicitamente dalla soluzione di un problema di ottimizzazione. In molti casi di interesse pratico si possono tuttavia dimostrare proprietà utili della funzione di ricorso, in particolare la **convessità**.

Poiché le variabili di secondo stadio sono funzioni in uno spazio infinito-dimensionale, una strategia comune consiste nella discretizzazione tramite campionamento. Si genera un **albero di scenario** (*fan*) in cui l'evento ω_s corrisponde alla realizzazione dello scenario $s \in S$, con S insieme degli scenari. Se gli scenari sono **campionati via Monte Carlo semplice**, le probabilità sono uniformi: $\pi_s = 1/|S|$ (sono possibili anche metodi più sofisticati di generazione scenari).

Modello deterministico equivalente (LP a grande scala) La discretizzazione conduce a

$$\begin{aligned} \min\ & c^T x + \sum_{s\in S} \pi_s \, (q^s)^T y^s \\ \text{s.t.}\ & Ax = b \\ & W y_s + T_s x = h_s, \quad s \in S \\ & x,\ y_s \geq 0. \end{aligned}$$

Fattibilità del secondo stadio e concetti di ricorso completo Una questione centrale è stabilire se **il secondo stadio sia fattibile per ogni scelta delle variabili di primo stadio** e per ogni realizzazione delle variabili aleatorie. Occorre restringere l'insieme delle decisioni *here-and-now* al dominio in cui il secondo stadio è fattibile (equivalentemente, la funzione di ricorso è limitata; come usuale, a un problema infeattibile si associa costo infinito). Riscrivendo i vincoli di collegamento come

$$W y_s = h_s - T_s x, \quad s \in S,$$

il **secondo stadio è fattibile quando il termine noto può essere espresso come combinazione conica delle colonne di** W. In tal caso si parla di **complete recourse**. Se il ricorso completo non vale per decisioni arbitrarie di primo stadio, ma vale per le decisioni che soddisfano i vincoli del primo stadio, si parla di **relatively complete recourse**.

Nei problemi di business è spesso possibile introdurre flessibilità tramite penalità, in modo da evitare infeattibilità al secondo stadio.

1.6.1 The plant location model

Descrizione del problema e rete bipartita Il classico **plant location model** è il più semplice problema di **network design**. Si considera una rete bipartita con un insieme P di nodi sorgente potenziali e un insieme $\mathcal{D}$ di nodi domanda. I nodi sorgente rappresentano impianti produttivi che possono essere aperti oppure no; i nodi domanda possono rappresentare magazzini regionali o centri retail. In ciascun nodo domanda si realizza una domanda aleatoria $D_j(\omega)$, $j \in \mathcal{D}$, che va soddisfatta al costo minimo. Sono necessari i seguenti dati

- Per ogni $i \in P$, un **costo fisso di apertura** f_i e una **capacità** u_i.
- Per ogni $j \in \mathcal{D}$ e **scenario** $s \in S$, una domanda d_j^s (discretizzazione della domanda aleatoria).
- Per ogni arco (i, j), un **costo unitario di trasporto** c_{ij} (costi variabili lineari).

Il **problema è naturalmente a due stadi**: le *decisioni di apertura impianti* devono essere prese sotto incertezza, mentre le *decisioni di trasporto* possono essere rimandate a dopo l'osservazione della domanda. Si introducono le variabili

$$y_i = \begin{cases} 1 & \text{se il nodo sorgente } i \in P \text{ è aperto} \\ 0 & \text{altrimenti} \end{cases} \qquad \text{e} \qquad x_{ij}^s \geq 0,$$

dove x_{ij}^s è il flusso da $i \in P$ a $j \in \mathcal{D}$ nello scenario $s \in S$. Le variabili di localizzazione y_i sono **di primo stadio** (design), mentre i flussi x_{ij}^s sono **di secondo stadio** (controllo).

Si ottiene il seguente **modello MILP stocastico a due stadi con ricorso**

$$\begin{aligned}
\min \quad & \sum_{i\in\mathcal{P}} f_i y_i + \sum_{s\in S} \pi^s \left(\sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{D}} c_{ij} x_{ij}^s \right) \\
\text{s.t.} \quad & \sum_{i\in\mathcal{P}} x_{ij}^s = d_j^s, && \forall s \in S,\ \forall j \in \mathcal{D} \\
& \sum_{j\in\mathcal{D}} x_{ij}^s \le R_i y_i, && \forall s \in S,\ \forall i \in \mathcal{P} \\
& x_{ij}^s \ge 0, y_i \in \{0,1\}.
\end{aligned}$$

Scenari estremi e necessità di formulazioni elastiche È opportuno chiedersi *cosa accade se viene incluso uno scenario estremo con domanda molto elevata*. Potrebbe non essere nemmeno possibile soddisfare tale domanda; inoltre, la pianificazione della capacità verrebbe guidata da scenari estremi ma molto improbabili, producendo soluzioni eccessivamente costose. In questi casi conviene ricorrere a formulazioni elastiche, permettendo **violazioni penalizzate dei vincoli**.

Modello elastico con domanda non soddisfatta penalizzata Sia $z_j^s \ge 0$ la **quantità di domanda non soddisfatta** nel nodo j nello scenario S. Tali variabili entrano nell'obiettivo moltiplicate per un coefficiente di penalità β_j. Il modello diventa

$$\begin{aligned}
\min \quad & \sum_{i\in\mathcal{P}} f_i y_i + \sum_{s\in S} \pi^s \left(\sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{D}} c_{ij} x_{ij}^s \right) + \sum_{s\in S} \pi^s \left(\sum_{j\in\mathcal{D}} \beta_j z_j^s \right) \\
\text{s.t.} \quad & \sum_{i\in\mathcal{P}} x_{ij}^s + z_j^s = d_j^s, && \forall s \in S,\ \forall j \in \mathcal{D} \\
& \sum_{j\in\mathcal{D}} x_{ij}^s \le R_i y_i, && \forall s \in S,\ \forall i \in \mathcal{P} \\
& x_{ij}^s \ge 0,\ z_j^s \ge 0, y_i \in \{0,1\}.
\end{aligned}$$

La **quantificazione delle penalità** β_j dipende dal significato di z_j^s: può rappresentare *domanda effettivamente non servita* (con penalità differenziate per priorità di mercato) oppure *domanda soddisfatta tramite fornitori terzi* (con penalità pari al costo effettivo di tale opzione).

Osservazioni di modellazione Il modello considerato è solo un primo passo verso formulazioni più realistiche (costi di trasporto non lineari, più item). Inoltre si trascura la dimensione temporale e l'uso di scorte per gestire variabilità parzialmente prevedibile della domanda, che porterebbero a un modello multistadio più impegnativo. L'output rilevante del modello rimane la **scelta del network design**.

1.6.2 Modello del newsvendor a due stadi

Motivazione e struttura informativa Si consideri una **versione a due stadi e multi-item del newsvendor problem**, soggetta a vincoli di capacità. La struttura informativa può essere rappresentata da un albero a tre livelli: al nodo radice si decide il primo lotto; nei nodi intermedi si decide un secondo lotto sotto incertezza ridotta; nei nodi foglia si osservano le vendite effettive e si contabilizzano costi di overage/underage. Ogni nodo foglia $n \in N_2$ ha esattamente un predecessore (antecedente) $\alpha(n) \in N_1$. Il nodo radice 0 è l'antecedente diretto di ogni nodo in N_1. Le decisioni dell'ultimo livello sono in realtà variabili fittizie di contabilizzazione (surplus/shortfall penalizzati), non variabili di controllo o design. Sono necessari i seguenti insiemi e parametri

- I: insieme degli item (SKU).

- K_1 e K_2: capacità massime (numero massimo di item producibili) nel primo e nel secondo run produttivo.
- L_i, $i \in I$: lotti minimi. Le variabili di lotto sono semicontinue: si può non produrre un item, ma se lo si produce esiste un minimo economicamente giustificato.
- Domanda d_i^n per ciascun $i \in I$ e foglia $n \in N_2$, con probabilità associata π^n.
- Coefficienti di penalità di overage c_i^o e underage c_i^u per ciascun $i \in I$.

Si introducono le **variabili decisionali**

- x_i^n: quantità prodotta dell'item i nel nodo $n \in \{0\} \cup N_1$, intera non negativa.
- Variabili binarie $z_i^n \in \{0,1\}$ per rappresentare la semicontinuità.
- Variabili di contabilizzazione u_i^n (underage) e v_i^n (overage) ai nodi foglia $n \in N_2$.

La formulazione **nel MILP stocastico** è

$$
\begin{aligned}
\min \quad & \sum_{n \in \mathcal{N}_2} \pi^n \left[\sum_{i \in \mathcal{I}} \left(c_i^o \, o_i^n + c_i^u \, u_i^n \right) \right] \\
\text{s.t.} \quad & \sum_{i \in \mathcal{I}} x_i^0 \le K_1 \\
& x_i^0 \ge m_i \delta_i^0, \qquad x_i^0 \le K_1 \delta_i^0 && i \in \mathcal{I} \\
& \sum_{i \in \mathcal{I}} x_i^n \le K_2 && n \in \mathcal{N}_1 \\
& x_i^n \ge m_i \delta_i^n, \qquad x_i^n \le K_2 \delta_i^n && i \in \mathcal{I},\ n \in \mathcal{N}_1 \\
& x_i^0 + x_i^{a(n)} = d_i^n + o_i^n - u_i^n && i \in \mathcal{I},\ n \in \mathcal{N}_2 \\
& x_i^n \in \mathbb{Z}_+, \qquad \delta_i^n \in \{0,1\} && i \in \mathcal{I},\ n \in \{0\} \cup \mathcal{N}_1 \\
& u_i^n,\ o_i^n \ge 0 && i \in \mathcal{I},\ n \in \mathcal{N}_2.
\end{aligned}
$$

1.6.3 Programmazione lineare stocastica multistadio con ricorso

Generalizzazione ai modelli multistadio Le *formulazioni multistadio sorgono come generalizzazione dei modelli a due stadi.* Concettualmente, **si annidano funzioni di ricorso corrispondenti ai diversi stadi decisionali**. In questo caso, le decisioni future possono dipendere dall'intera storia del processo stocastico dei fattori di rischio. Si introduce la notazione

$$
\tilde{\xi}_{[t]} = (\tilde{\xi}_1, \tilde{\xi}_2, \ldots, \tilde{\xi}_t), \qquad \tilde{\xi}_{[1]} = \tilde{\xi}_1.
$$

La decisione *here-and-now* è $x_0 \in X_0$ e si risolve

$$
\min_{x_0 \in X_0} \left\{ f_0(x_0) + \mathbb{E}\big[Q(x_0, \tilde{\xi}_1) \big] \right\}.
$$

Il termine $Q(x_0, \xi_1)$ è definito ricorsivamente come

$$
Q(x_0, \xi_1) = \min_{x_1 \in X_1(x_0, \xi_1)} \left\{ f_1(x_1, \xi_1) + \mathbb{E}\Big[Q\big(x_1, \tilde{\xi}_{[2]}\big) \mid \tilde{\xi}_1 = \xi_1 \Big] \right\}.
$$

Allo stesso modo, per $t = 2, \ldots, T$, si ha

$$
Q_t\Big(x_{t-1}, \tilde{\xi}_{[t]}\Big) = \min_{x_t \in X_t(x_{t-1}, \tilde{\xi}_{[t]})} \left\{ f_t\big(x_t, \xi_{[t]}\big) + \mathbb{E}\left[Q_{t+1}\Big(x_t, \tilde{\xi}_{[t+1]}\Big) \;\middle|\; \tilde{\xi}_{[t]} = \xi_{[t]} \right] \right\}.
$$

Possiamo definire la funzione di ricorso

$$Q_{t+1}\big(x_t, \xi_{[t]}\big) \;\doteq\; \mathbb{E}\left[Q_{t+1}\Big(x_t, \tilde{\xi}_{[t+1]}\Big) \;\middle|\; \tilde{\xi}_{[t]} = \xi_{[t]}\right],$$

e riscrivere l'equazione precedente come

$$Q_t\big(x_{t-1}, \xi_{[t]}\big) = \min_{x_t \in X_t(x_{t-1}, \xi_{[t]})} \big\{ f_t\big(x_t, \xi_{[t]}\big) + Q_{t+1}\big(x_t, \xi_{[t]}\big) \big\}.$$

Nel problema dell'ultimo stadio, per $t = T$, si può semplicemente porre $Q_T\big(x_T, \xi_{[T]}\big) \equiv 0$.

Una formulazione alternativa,si ottiene enfatizzando che le decisioni di ricorso sono in realtà funzioni (*non anticipative*) del cammino campionario osservato fino a quel momento, $x_t(\tilde{\xi}_{[t]})$. Questo conduce alla formulazione funzionale

$$\min_{x_0,\, x_1(\cdot), \ldots, x_T(\cdot)} \mathbb{E}\Big[f_0(x_0) + f_1\Big(x_1(\tilde{\xi}_{[1]}), \tilde{\xi}_1\Big) + \cdots + f_T\Big(x_T(\tilde{\xi}_{[T]}), \tilde{\xi}_T\Big) \Big]$$

$$\begin{aligned} \text{s.t.} \quad & x_0 \in X_0, \\ & x_t(\xi_{[t]}),\, \xi_t \in X_t\big(x_{t-1}(\xi_{[t-1]}), \xi_{t-1}\big), \qquad t = 1, \ldots, T. \end{aligned}$$

1.6.4 Un modello di pianificazione finanziaria multistadio con costi di transazione proporzionali

Asset–liability management su albero di scenario Si formula un modello di **asset–liability management** in cui si scambia un insieme di strumenti finanziari (asset) per far fronte a un flusso di passività (liabilities). L'incertezza è rappresentata mediante un albero di scenario.

- L_n è la liability da soddisfare nel nodo $n \in N$.
- Non si considerano nuovi apporti di cassa lungo il percorso; l'unico modo per generare cassa è vendere asset.
- Gli asset sono indicizzati da $i \in I$ e P_i^n è il prezzo dell'asset i nel nodo n.
- Si considerano costi di transazione proporzionali (lineari): per acquistare o vendere si paga una percentuale c del valore scambiato, sia in acquisto sia in vendita.

La radice è il nodo n_0, in cui si effettua la prima allocazione considerando le dotazioni iniziali $\bar{h}_i^{n_0}$. L'insieme delle foglie è S; i nodi di trading intermedi sono $T = N \setminus \big(\{n_0\} \cup S\big)$. L'obiettivo è massimizzare l'utilità attesa della ricchezza terminale. Si introducono le variabili decisionali:

- z_i^n: quantità di asset i acquistata nel nodo n.
- y_i^n: quantità di asset i venduta nel nodo n.
- x_i^n: quantità detenuta di asset i nel nodo n dopo il ribilanciamento.
- W^ℓ: ricchezza terminale nella foglia $s \in S$.

Le variabili z_i^n, y_i^n, x_i^n sono definite per $n \in N \setminus S$ (nessun ribilanciamento alle foglie). Sia $U(\cdot)$ l'utilità non lineare della ricchezza.

$$
\begin{aligned}
\max \quad & \sum_{s\in S} \pi^s\, u(W^s) \\
\text{s.t.} \quad & x_i^{n_0} = \bar{h}_i^{n_0} + z_i^{n_0} - y_i^{n_0}, && i \in \mathcal{I} \\
& x_i^n = x_i^{a(n)} + z_i^n - y_i^n, && i \in \mathcal{I},\ n \in \mathcal{T} \\
& (1-c)\sum_{i\in\mathcal{I}} P_i^n y_i^n - (1+c)\sum_{i\in\mathcal{I}} P_i^n z_i^n = L^n, && n \in \mathcal{N}\setminus S \\
& W^s = (1-c)\sum_{i\in\mathcal{I}} P_i^s x_i^{a(s)} - L^s, && s \in S \\
& x_i^n,\ z_i^n,\ y_i^n,\ W^s \geq 0.
\end{aligned}
$$

Osservazioni operative In pratica il modello verrebbe risolto ripetutamente con una logica *rolling horizon*; il ruolo dell'utilità terminale è evitare effetti distorsivi di fine orizzonte, imponendo che il portafoglio sia in una buona posizione alla fine dell'orizzonte di pianificazione. In un modello più realistico si dovrebbero includere eventuali contributi in ingresso e shortfall penalizzati rispetto alle liabilities.

1.7 Modelli multiperiodali e multistadio

Esplosione dell'albero di scenario nei modelli multistadio Un problema chiaro nei modelli di programmazione stocastica **multistadio** è la possibile **esplosione dell'albero di scenario**. Il numero di nodi necessario per rappresentare l'incertezza può diventare un fattore fortemente limitante. Questo problema è meno rilevante per una struttura ad albero **lineare**. È però importante osservare che una struttura ad albero lineare **non corrisponde a un modello multistadio**, bensì a un **modello a due stadi multiperiodale**.

Multiperiodo non significa multistage È possibile avere un modello multiperiodale ma **statico** se tutte le decisioni sono prese **here-and-now** e poi implementate nei periodi successivi senza adattamento al flusso informativo. Al contrario, **multistage** implica **adattamento** delle decisioni nel tempo, soggetto a vincoli di **non-anticipatività** delle politiche decisionali.

Caso particolare del two-stage multiperiod La struttura è peculiare: è **a due stadi**, ma coinvolge **più periodi**. Apparentemente viola la non-anticipatività, perché dopo la prima diramazione successiva alla radice sarebbe possibile prevedere perfettamente il cammino futuro. Tuttavia, una tale struttura può avere senso quando:

- Si devono prendere **decisioni strutturali o di design here-and-now** che non possono essere adattate.
- Segue un flusso di decisioni di controllo nel tempo, per le quali non si può trarre vantaggio dalla struttura lineare dell'albero.

Il modello di pianificazione energetica della sezione successiva illustra un caso di questo tipo.

1.7.1 Un modello multiperiodo a due stadi: Unit commitment

Domanda aleatoria come processo discreto nel tempo Si consideri un modello stocastico di **unit commitment**, problema di pianificazione energetica in cui l'incertezza della domanda è modellata come processo stocastico a tempo discreto

$$
\xi(\omega) = \big(d_1(\omega), \dots, d_T(\omega)\big), \qquad t \in \{1, \dots, T\},\ \omega \in \Omega,
$$

dove T è la lunghezza dell'orizzonte di pianificazione e Ω è l'insieme (finito) dei cammini campionari. Si denoti con π^ω la probabilità dello scenario ω.

Classi di generatori e struttura dei costi Si dispone di un insieme di I classi di unità generatrici. Una unità della classe i (con $i = 1, \dots, I$) può produrre in un intervallo $[m_i, M_i]$ quando è attiva. Per ciascuna classe i sono disponibili a_i unità.

Si consideri un insieme di I classi di unità di generazione, che possono fornire un livello di output nell'intervallo $[m_i, M_i]$, $i \in [I]$, quando sono attive. Per ciascuna classe i sono disponibili a_i unità. Anche in questo caso, la potenza erogata è una variabile decisionale *semicontinua*, che può assumere valore nullo, ma che presenta un livello minimo m_i per un'unità attiva. Se un'unità di generazione della classe i è attiva al tempo t, si sostiene un costo E_i per mantenerla al livello minimo m_i, e un costo variabile lineare C_i per ogni unità addizionale di potenza erogata. Si sostiene inoltre un costo di avviamento s_i per l'accensione di un'unità della classe i.

Viene introdotta una variabile decisionale intera non negativa u_{it} che denota il numero totale di unità della classe i attive durante l'intervallo temporale t. Data la struttura dei costi, non risulta necessario rappresentare ciascuna singola unità. Si considera inoltre una variabile intera non negativa s_{it} che rappresenta il numero di unità della classe i che vengono accese all'inizio dell'intervallo temporale t. Tali variabili costituiscono decisioni di *primo stadio*. Al contrario, la variabile non negativa q_{it}^ω, che rappresenta l'output totale della classe i durante l'intervallo temporale t, è una variabile decisionale di *secondo stadio*, indicizzata per scenario. Il **modello** può essere formulato come segue

$$
\begin{aligned}
\min \quad & \sum_{i\in[I],\, t\in[T]} (E_i u_{it} + F_i s_{it}) + \sum_{\omega\in\Omega} \pi^\omega \left[\sum_{i\in[I],\, t\in[T]} C_i \left(q_{it}^\omega - m_i u_{it}\right) \right] & \\
\text{s.t.} \quad & \sum_{i\in[I]} q_{it}^\omega \geq d_t(\omega), & t \in [T],\ \omega \in \Omega \\
& m_i u_{it} \leq q_{it}^\omega \leq M_i u_{it}, & i \in [I],\ t \in [T],\ \omega \in \Omega \\
& s_{it} \geq u_{it} - u_{i,t-1}, & i \in [I],\ t \in [T] \\
& u_{it} \leq a_i, & i \in [I],\ t \in [T] \\
& u_{it} \in \mathbb{Z}_+,\ s_{it} \in \mathbb{Z}_+,\ q_{it}^\omega \geq 0, & i \in [I],\ t \in [T],\ \omega \in \Omega.
\end{aligned}
$$

Osservazioni critiche sul modello Si notano due aspetti rilevanti

- **Fattibilità e scenari estremi.** Se il vincolo di domanda è imposto come vincolo rigido in ogni scenario, la soluzione può diventare molto costosa e guidata da pochi scenari improbabili con picchi di domanda elevati. Come nel plant location model, può essere opportuno introdurre una formulazione elastica, ad esempio includendo una unità "di ultima istanza" con capacità illimitata (fornitore esterno) o una penalità per domanda non soddisfatta.
- **Flusso informativo.** Il modello è multiperiodale ma a due stadi: dopo aver osservato la prima domanda al tempo $t = 1$, si assume di conoscere tutte le domande per $t \in \{2, \dots, T\}$. Questo non è realistico e si potrebbe pensare a un modello multistadio con vincoli di non-anticipatività. Tuttavia, qui non c'è modo di sfruttare tale informazione, perché le variabili di design sono *here-and-now*. Inoltre, nel modello non esiste uno stato energetico (l'energia non è immagazzinabile), e quindi i periodi sono indipendenti tra loro in termini di potenza erogata.

1.8 Generazione di scenari e stabilità nella programmazione stocastica

Ruolo centrale dell'albero di scenario Nei modelli di programmazione stocastica **multistadio**, l'elemento chiave è l'**albero di scenario**: la qualità della soluzione dipende in modo critico da quanto

bene l'albero rappresenta l'incertezza. In linea di principio si potrebbe campionare un modello dinamico dei fattori di rischio tramite **simulazione Monte Carlo**, ma sorgono ulteriori difficoltà.

Complicazioni principali

- La **dimensione del campione** deve restare contenuta, perché si associano variabili decisionali agli stati e risolvere un problema di ottimizzazione stocastica (non solo calcolare una media campionaria) può essere computazionalmente oneroso.
- Le decisioni devono essere **non-anticipative**: ciò richiede una struttura ad albero soggetta a crescita **esponenziale**. Ad esempio, con 100 rami per nodo si ottengono un milione di scenari dopo tre passi.
- Non si sta solo stimando un valore atteso, ma si stanno **prendendo decisioni** risolvendo un problema di ottimizzazione: qualunque **bias** o incoerenza nell'albero verrà sfruttata dal solver, producendo una soluzione magari buona *in-sample* ma debole *out-of-sample*.

Qualunque strategia di generazione scenari si scelga, è quindi importante controllare la **stabilità** della soluzione risultante.

Due classi di approcci alla generazione di scenari Si distinguono due approcci fondamentali

- **Generazione stocastica di scenari.** Basata su campionamento casuale **Monte Carlo** e quindi sulla "forza bruta" di una grande numerosità campionaria. È più gestibile nei modelli **a due stadi** che nei multistadio; le prestazioni possono essere migliorate con tecniche di **riduzione della varianza**, ad esempio *importance sampling*.
- **Generazione deterministica di scenari.** Invece della forza bruta, si selezionano scenari in modo "intelligente" tramite metodi quali **quadratura gaussiana**, **sequenze a bassa discrepanza** (ad es. **Sobol**) o procedure di generazione **ottimizzata**.

Interpretazione come integrazione numerica (quadratura) La generazione scenari può essere vista come un problema di **integrazione numerica**. Se $f(x, \tilde{\xi})$ dipende dalle decisioni x e da variabili aleatorie $\tilde{\xi}$ con densità congiunta $h(\xi)$, allora

$$F(x) = \int f(x,\xi)\, h(\xi)\, d\xi \approx \frac{1}{S} \sum_{s=1}^{S} f(x, \xi_s).$$

In alternativa, si può usare un insieme (anche deterministico) di punti con pesi $\{(\pi_s, \xi_s)\}$:

$$F(x) \approx \sum_{s=1}^{S'} \pi_s\, f(x, \xi_s), \qquad S' < S,$$

interpretando le coppie (π_s, ξ_s) come una **distribuzione discreta** che approssima quella continua.

Moment/property matching e alternative Un'idea comune è scegliere scenari e probabilità in modo da riprodurre (approssimativamente) alcune **proprietà** della distribuzione, ad esempio i **momenti** (*moment matching*). Questo approccio è stato criticato perché è possibile costruire distribuzioni diverse che condividono i primi momenti. Un'alternativa è utilizzare **metriche di distanza** tra distribuzioni: fissata una topologia dell'albero, si cercano valori e probabilità che minimizzano una distanza rispetto alla distribuzione "vera", oppure si parte da un grande insieme di scenari e si applica **scenario reduction** per ottenere un albero più gestibile. D'altra parte, spesso ciò che conta non è la distanza tra distribuzioni, ma la **qualità della soluzione** (ad es. in un modello mean–variance può essere sufficiente approssimare bene i primi due momenti).

Linee guida pratiche per progettare l'albero

- Se l'obiettivo è una decisione robusta di primo stadio, conviene spesso usare un branching **ricco al primo stadio** e più **limitato negli stadi successivi** (in finanza, imponendo che l'albero sia **arbitrage-free**).
- Si può ridurre la complessità usando **passi temporali non uniformi** (più fitti all'inizio, più radi in seguito).
- Si può limitare il numero di stadi **aumentando l'obiettivo** con un termine che valuta la qualità dello **terminale**, riducendo decisioni miopi e permettendo alberi più piccoli.

1.8.1 In– and out–of–sample stability

Perché serve controllare la stabilità Qualunque strategia di generazione scenari venga impiegata, è importante verificare la **stabilità** della soluzione. In pratica, l'ottimizzazione usa un albero approssimato e quindi occorre capire quanto l'output dipenda da come l'albero è stato generato.

Problema "esatto" e problema approssimato Il problema stocastico "esatto" può essere scritto come

$$\min_{x \in X} \ \mathbb{E}_{\mathbb{P}}\Big[f(x, \tilde{\xi})\Big],$$

dove P è la **misura vera** dei fattori di rischio. In pratica si risolve un problema basato su un albero di scenario approssimato T

$$\min_{x \in S} \ \widehat{f}(x; T),$$

dove $\widehat{f}$ rappresenta una stima del valore atteso indotta dall'albero.

Stabilità del valore obiettivo Campionando (o costruendo) alberi diversi si ottengono soluzioni ottime diverse e valori obiettivo diversi. Spesso è più informativo controllare la stabilità del **valore dell'obiettivo** (piuttosto che della soluzione), perché quando l'obiettivo è piatto soluzioni anche molto differenti possono avere prestazioni simili.

In-sample stability La **stabilità *in-sample*** richiede che, generando due alberi T_1 e T_2, i valori obiettivo ottenuti non differiscano troppo: $\widehat{f}(x_1^*; T_1) \approx \widehat{f}(x_2^*; T_2)$. Se l'albero è generato in modo deterministico, si può comunque confrontare la stabilità variando leggermente la struttura di branching per verificare che sia sufficientemente ricca.

Out-of-sample stability La **stabilità *out-of-sample*** confronta la performance reale attesa delle soluzioni sotto la distribuzione vera: $\mathbb{E}_{\mathbb{P}}\Big[f(x_1^*, \tilde{\xi})\Big] \approx \mathbb{E}_{\mathbb{P}}\Big[f(x_2^*, \tilde{\xi})\Big]$. Se gli alberi sono affidabili, le prestazioni reali non dovrebbero cambiare in modo significativo.

Valutazione out-of-sample: due stadi vs multistadio Nei modelli a **due stadi** la verifica out-of-sample è relativamente semplice: si fissa la soluzione di primo stadio e si risolvono molti problemi di secondo stadio corrispondenti a diverse realizzazioni di $\tilde{\xi}$. Nei modelli **multistadio**, invece, una valutazione realistica richiede tipicamente una simulazione **rolling horizon**, che è computazionalmente costosa.

Rolling horizon e shrinking horizon

- **Sliding/rolling horizon**: si risolve un problema con H stadi, si applica la prima decisione, si osserva una realizzazione out-of-sample e si risolve di nuovo un problema con H stadi a partire dal periodo successivo.

- **Shrinking horizon**: si risolve con H stadi, si applica la prima decisione, si osserva una realizzazione e poi si risolve con $H-1$ stadi, riducendo progressivamente l'orizzonte. Lo shrinking horizon è più economico quando l'orizzonte di valutazione è ben definito.

Test economico alternativo: scambio degli alberi Un controllo più economico consiste nel generare soluzioni da due alberi T_1 e T_2 e **valutarle scambiando gli alberi**, verificando se $\widehat{f}(x_1^*; T_2) \approx \widehat{f}(x_2^*; T_1)$.

Osservazione finale: limite operativo dello stochastic programming La necessità di valutazione out-of-sample evidenzia che la programmazione stocastica produce decisioni esplicite **associate ai nodi dell'albero**; non fornisce direttamente una politica generale facilmente applicabile quando l'incertezza si realizza fuori dagli scenari previsti. Approcci come **programmazione dinamica** e **decision rules** restituiscono invece decisioni in **feedback**, più adatte alla simulazione e alla valutazione delle politiche.

1.9 Convessità nei modelli di programmazione stocastica

Vincoli probabilistici congiunti Si consideri un vincolo probabilistico congiunto della forma $\mathbb{P}\big(\{\omega \mid \mathbf{g}(\mathbf{x}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0}\}\big) \geq 1-\alpha$, dove $\mathbf{g}$ è una funzione vettoriale. Un punto $\hat{\mathbf{x}}$ è ammissibile se l'insieme $\mathbb{S}(\hat{\mathbf{x}}) = \{\omega \mid \mathbf{g}(\hat{\mathbf{x}}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0}\}$ ha misura di probabilità almeno pari a $1-\alpha$.

Sia $\mathcal{F}$ il campo di tutti gli eventi e sia $\mathcal{G} \subseteq \mathcal{F}$ la collezione degli eventi con probabilità almeno $1-\alpha$. Allora $\hat{x}$ è *ammissibile* se esiste almeno un evento $G \in \mathcal{G}$ tale che $\mathbf{g}(\hat{\mathbf{x}}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0} \quad \forall\, \omega \in G$, ossia

$$\hat{\mathbf{x}} \in \bigcap_{\omega \in G} \{\mathbf{x} \mid \mathbf{g}(\mathbf{x}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0}\}\,.$$

Struttura dell'insieme ammissibile L'**insieme ammissibile** complessivo può quindi essere scritto come

$$X = \bigcup_{G \in \mathcal{G}} \bigcap_{\omega \in G} \{\mathbf{x} \mid \mathbf{g}(\mathbf{x}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0}\}\,.$$

Anche se per ogni ω l'insieme $\{\mathbf{x} \mid \mathbf{g}(\mathbf{x}; \boldsymbol{\xi}(\omega)) \leq \mathbf{0}\}$ è convesso, non ci si può aspettare che l'insieme X sia convesso in generale, poiché l'unione di insiemi convessi non è necessariamente convessa. La convessità del problema complessivo dipende dal tipo di distribuzione (continua o discreta), dalla sua funzione di distribuzione cumulativa, dalla natura dei vincoli (congiunti o disgiunti) e dalla struttura delle funzioni in $\mathbf{g}$ (lineari, convesse/concave o arbitrarie).

Vincolo probabilistico lineare con distribuzione normale Si consideri un vincolo lineare casuale $\mathbf{a}^T\mathbf{x} \leq b$, dove $\mathbf{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, e si richieda $\mathbb{P}\{\mathbf{a}^T\mathbf{x} \leq b\} \geq \eta$. Per un dato vettore $\mathbf{x}$, la variabile casuale $\mathbf{a}^T\mathbf{x}$ è distribuita come $\mathbf{a}^T\mathbf{x} \sim \mathcal{N}(\nu, \sigma^2)$, dove $\nu = \boldsymbol{\mu}^T\mathbf{x}$ e $\sigma^2 = \mathbf{x}^T\boldsymbol{\Sigma}\mathbf{x}$.

Usando la funzione di distribuzione cumulativa della normale standard $\Phi(\cdot)$, il vincolo può essere riscritto come

$$\mathbb{P}\left\{\frac{\mathbf{a}^T\mathbf{x}-\nu}{\sigma} \leq \frac{b-\nu}{\sigma}\right\} = \Phi\left(\frac{b-\nu}{\sigma}\right) \geq \eta \iff \frac{b-\nu}{\sigma} \geq \Phi^{-1}(\eta).$$

Assumendo $\eta > 0.5$, quindi $\Phi^{-1}(\eta) > 0$, il vincolo diventa

$$\boldsymbol{\mu}^T\mathbf{x} + \Phi^{-1}(\eta)\,\|\boldsymbol{\Sigma}^{1/2}\mathbf{x}\|_2 \leq b,$$

che è un vincolo conico del secondo ordine. Ne segue che un problema di programmazione lineare con vincoli probabilistici disgiunti di questo tipo è un **problema convesso di tipo SOCP**.

Convessità della funzione di ricorso Si consideri ora la funzione di ricorso $Q(\mathbf{x}) = \mathbb{E}[Q(\mathbf{x}; \tilde{\boldsymbol{\xi}})]$ nel caso di costo di ricorso deterministico, con

$$Q(\mathbf{x}; \boldsymbol{\xi}) = \min_{\mathbf{y}} \left\{ \mathbf{q}^T \mathbf{y} \mid \mathbf{W}\mathbf{y} = \mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x},\ \mathbf{y} \geq \mathbf{0} \right\}.$$

Il dominio effettivo di $Q(\mathbf{x})$ è costituito dai vettori $\mathbf{x}$ ammissibili per i vincoli di primo stadio tali che $Q(\mathbf{x}) < +\infty$, ossia per cui il problema di secondo stadio è (quasi sicuramente) ammissibile.

Per **dualità della programmazione lineare** si ottiene

$$Q(\mathbf{x}; \boldsymbol{\xi}) = \max_{\boldsymbol{\pi}} \left\{ [\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}]^T \boldsymbol{\pi} \mid \mathbf{W}^T \boldsymbol{\pi} \leq \mathbf{q} \right\}.$$

Si denoti con $\Pi = \{\boldsymbol{\pi} \mid \mathbf{W}^T \boldsymbol{\pi} \leq \mathbf{q}\}$ l'**insieme ammissibile del duale**, che è non casuale poiché $\mathbf{q}$ è deterministico.

Dimostrazione della convessità Siano $\mathbf{x}_1, \mathbf{x}_2 \in D$, $\lambda \in [0,1]$, e $\mathbf{x}_\lambda = \lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2$. Allora Si considerino $\mathbf{x}^1, \mathbf{x}^2 \in \mathcal{D}$, $\lambda \in [0,1]$, e $\mathbf{x}_\lambda = \lambda \mathbf{x}^1 + (1-\lambda)\mathbf{x}^2$.

$$\begin{aligned}
Q(\mathbf{x}_\lambda) &\overset{\text{def}}{=} \int Q(\mathbf{x}_\lambda, \boldsymbol{\xi})\, dP \\
&= \int \max_{\boldsymbol{\pi} \in \Pi} \left\{ \left[\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}_\lambda\right]^T \boldsymbol{\pi} \right\} dP \\
&= \int \max_{\boldsymbol{\pi} \in \Pi} \left\{ \lambda \left[\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}^1\right]^T \boldsymbol{\pi} + (1-\lambda) \left[\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}^2\right]^T \boldsymbol{\pi} \right\} dP \\
&\leq \lambda \int \max_{\boldsymbol{\pi} \in \Pi} \left\{ \left[\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}^1\right]^T \boldsymbol{\pi} \right\} dP + (1-\lambda) \int \max_{\boldsymbol{\pi} \in \Pi} \left\{ \left[\mathbf{h}(\boldsymbol{\xi}) - \mathbf{T}(\boldsymbol{\xi})\mathbf{x}^2\right]^T \boldsymbol{\pi} \right\} dP \\
&= \lambda Q(\mathbf{x}^1) + (1-\lambda) Q(\mathbf{x}^2).
\end{aligned}$$

Pertanto la funzione di ricorso è convessa.

Osservazioni finali La dimostrazione precedente risulta particolarmente semplice poiché l'**insieme ammissibile del problema duale è non casuale**. In un contesto più generale, tuttavia, la **convessità della funzione di ricorso** può essere dimostrata anche senza questa assunzione. In particolare, la funzione di ricorso è **tipicamente continua** quando le variabili aleatorie seguono **distribuzioni continue**, mentre nel caso di **distribuzioni discrete** essa assume una struttura **poliedrica**. In entrambi i casi è possibile applicare il **metodo dei piani di taglio di Kelley**. Inoltre, grazie alla **struttura del problema**, tali piani di taglio possono essere generati in modo efficiente tramite **strategie di decomposizione**.

2 CAP 2 – Elementi di complessità computazionale

Recap La **complessità computazionale** di un problema dipende dalla **dimensione dell'istanza**, intesa come *lunghezza della sua codifica*, e può crescere in modo **fattoriale**, **esponenziale** o **polinomiale**. Piccole modifiche nella struttura di un problema possono determinare variazioni drastiche della complessità, come accade nei problemi di scheduling $1//L_{\max}$ e $1/r_i/L_{\max}$.

È fondamentale distinguere tra **complessità di un algoritmo** e **complessità intrinseca di un problema**: la prima riguarda il numero di operazioni elementari richieste da una procedura di soluzione, mentre la seconda riguarda la difficoltà strutturale del problema stesso, indipendentemente dall'algoritmo utilizzato.

La **teoria della NP-completezza** fornisce un quadro concettuale per classificare i **problemi di decisione** in base alla loro difficoltà computazionale. In particolare

- La classe P contiene i problemi di decisione risolvibili in **tempo polinomiale**.

- La classe NP contiene i problemi di decisione per cui le istanze con risposta positiva sono **verificabili in tempo polinomiale** su calcolatori non deterministici.
- La classe NPH (NP-hard) raccoglie i problemi almeno difficili quanto tutti i problemi in NP.
- La classe NPC (NP-complete) contiene i problemi più difficili in NP, cioè quelli che sono *sia* in NP *sia* NP-hard.

Il concetto chiave che permette di confrontare la difficoltà dei problemi è quello di **riduzione in tempo polinomiale**: se un problema P può essere trasformato in un problema Q in tempo polinomiale, allora Q non può essere più facile di P. Tutti i problemi della classe NPC risultano **equivalenti in termini di complessità**.

Il **teorema di Cook** individua l'**innesco della catena delle riduzioni**, dimostrando che il **problema della soddisfacibilità Booleana** è NP-completo. A partire da esso, mediante riduzioni polinomiali successive, è possibile costruire un'ampia famiglia di problemi NP-completi, tra cui il problema **subset-sum**.

La NP-completezza della versione decisionale di $1/r_i/L_{\max}$ implica che il corrispondente **problema di ottimizzazione** è **NP-difficile**. In generale, dimostrare che una versione decisionale è NP-completa è sufficiente per concludere che il problema di ottimizzazione associato non ammette algoritmi polinomiali, salvo l'ipotesi $P = NP$.

Un aspetto cruciale è infine la **codifica dei dati**: la complessità va valutata rispetto alla rappresentazione binaria delle istanze. L'algoritmo di programmazione dinamica per il problema **knapsack** ha complessità $O(nB)$ ed è quindi **pseudo-polinomiale**, poiché dipende dal valore numerico dei dati e non dalla lunghezza della loro codifica. Questo chiarisce perché il knapsack rimane un problema NP-difficile nonostante l'esistenza di algoritmi efficienti in casi particolari.

Complessità di problemi e algoritmi Si vuole caratterizzare la *complessità* in funzione della *dimensione di un problema*. Occorre distinguere la complessità degli algoritmi da quella dei problemi. Nel caso di un algoritmo caratterizzato da un **numero finito di passi**, possiamo valutare (eventualmente nel caso peggiore) il **numero di operazioni elementari** in funzione della dimensione n del problema.

- **Algoritmi enumerativi**: algoritmi che considerano tutte le **permutazioni** di n oggetti. La loro complessità è $O(n!)$ ed è certamente non praticabile per valori di n anche moderatamente grandi.
- **Algoritmi di assegnamento esaustivo**: algoritmi che valutano tutti gli **assegnamenti possibili** di n variabili binarie. In questo caso la complessità cresce in modo esponenziale ed è pari a $O(2^n)$.
- **Algoritmi polinomiali**: un tipico esempio è rappresentato dagli **algoritmi di ordinamento** di n oggetti. Gli algoritmi più semplici hanno complessità $O(n^2)$, mentre algoritmi più efficienti raggiungono complessità $O(n \log_2 n)$.

Nel caso di **algoritmi iterativi** che generano una sequenza di soluzioni, si può cercare di caratterizzare la **velocità di convergenza** (es., lineare o quadratica).

Complessità intrinseca di un problema Una questione più sottile si pone quando si vuole caratterizzare la **complessità intrinseca di un problema**.

Per comprendere la natura della questione, consideriamo un semplice **problema di scheduling di n job su macchina singola**. Indichiamo con p_j il tempo necessario per il job $j = 1, \ldots, n$, e con d_j, $j = 1, \ldots, n$ la sua data di consegna (**due date**). La soluzione è una *sequenza di job*, ovvero una permutazione σ in cui $\sigma(k)$ è l'indice del job in posizione k. I **tempi di completamento** sono

$$\begin{aligned} C_{\sigma(1)} &= p_{\sigma(1)}, \\ C_{\sigma(k)} &= C_{\sigma(k-1)} + p_{\sigma(k)}, \quad k = 2, \ldots, n. \end{aligned}$$

Si vuole minimizzare la massima lateness, $L_{\max} \doteq \max_{j \in [n]} L_j$, dove $L_j \doteq C_j - d_j$. Tale problema viene indicato con la stringa $1//L_{\max}$.

Teorema (regola EDD – Earliest Due Date) Per il problema $1//L_{\max}$ esiste una **soluzione ottima** in cui $d_{\sigma(k)} \leq d_{\sigma(k+1)}$.

Dimostrazione Si consideri una soluzione ottima in cui, per due job consecutivi in sequenza, prima $i = \sigma(k)$ e poi $j = \sigma(k+1)$, risulti $d_i > d_j$. Si indichino con C_i e $C_j = C_i + p_j$ i due tempi di completamento nella soluzione corrente, per la quale si hanno due valori di lateness $L_i = C_i - d_i$ e $L_j = C_j - d_j$.
Scambiando i due job, si ottengono $C'_j < C_j$ e $C'_i = C_j$. Per il job j, che viene anticipato, vale $C'_j < C_j$ e quindi $L'_j < L_j$; di conseguenza, la lateness del job j migliora nella nuova soluzione. Per il job i, che viene spostato in avanti nella sequenza, risulta $L'_i = C'_i - d_i = C_j - d_i < C_j - d_j = L_j$, per cui la lateness del job i peggiora, ma non supera la precedente lateness del job j. La nuova soluzione risulta quindi migliore della precedente, in contraddizione con l'ipotesi di ottimalità.

Introduzione dei tempi di rilascio Si dispone dunque di un algoritmo di complessità polinomiale per il problema. **Cosa accade se il problema viene leggermente complicato, introducendo tempi di rilascio (release time o ready time) r_i, $i \in [n]$, per i job?** Per il problema $1/r_i/L_{\max}$ non sono noti algoritmi di complessità polinomiale ed è semplice costruire controesempi alla regola EDD (da adattare comunque alla disponibilità dei job).

Per il problema $1/r_i/L_{\max}$ esistono algoritmi **branch-and-bound** (quindi di complessità esponenziale). Non è noto un algoritmo di complessità polinomiale per questo problema di ottimizzazione combinatoria (e per molti altri), ma non è nemmeno stato dimostrato che tale algoritmo non possa esistere.

2.1 Caratterizzazione della complessità – classi P, NP, NPC e NPH

Teoria della NP-completezza Esiste una classe molto ampia di problemi di ottimizzazione per cui non sono disponibili *algoritmi di complessità polinomiale*. Tuttavia, la questione dell'esistenza o meno di un algoritmo di complessità polinomiale per essi rimane aperta. La **teoria della NP-completezza** permette di dare una risposta parziale alla questione, mostrando come questi problemi siano *equivalenti tra di loro*, nel senso che un algoritmo di complessità polinomiale per anche uno solo di essi fornirebbe un algoritmo di complessità polinomiale per tutti i problemi di una classe molto ampia.
Il fatto che decenni di ricerca sulla soluzione di tutti questi problemi non abbiano prodotto un algoritmo polinomiale **suggerisce che esso in effetti non esiste**.

Problemi di decisione e di ottimizzazione Occorre distinguere **problemi di decisione** e **problemi di ottimizzazione**. Esempi di problema di decisione sono i seguenti.

1. **Problema K_0 (subset sum)** Dati $n+1$ numeri interi positivi $a_1, a_2, \ldots, a_n$ e b, esiste un sottoinsieme $J \subseteq [n]$ tale che $\sum_{i \in J} a_i = b$?
2. **Problema LS_0** Dato un **problema di lot sizing multiprodotto**, con tempi e costi di setup, esiste una soluzione ammissibile rispetto al soddisfacimento della domanda e ai vincoli di capacità produttiva?

Data una specifica istanza di un *problema di decisione*, la risposta è *sì* oppure *no*. Dal punto di vista teorico, è più agevole trattare problemi di decisione, **ma è facile vedere il legame tra problemi di ottimizzazione e problemi di decisione**. Dato un problema di ottimizzazione $\min_{x \in S} f(x)$, **è possibile definirne una versione decisionale**, scegliendo un numero k e chiedendo se esiste $x \in S$ tale che $f(x) \leq k$, dove k è un numero intero. Si indica con PO il **problema di ottimizzazione**, e con PD il corrispondente **problema di decisione**.

Se si ha a disposizione un algoritmo efficiente per PO, allora è possibile risolvere in modo efficiente anche PD: basta risolvere il problema di ottimizzazione e verificare se $f(x^*) \leq k$. Questo permette di scrivere $PD \to PO$, nel senso che **il problema di decisione può essere ricondotto al problema di ottimizzazione**.

Non è detto che tale trasformazione sia conveniente, ma è possibile escludere che PO sia facile se PD è difficile, perché un ipotetico algoritmo efficiente per PO risolverebbe anche facilmente PD.

Quindi, *per dimostrare che un problema di ottimizzazione è difficile, può bastare dimostrare che è difficile il corrispondente problema di decisione*.

D'altro canto, un algoritmo di decisione efficiente potrebbe, in certi casi, essere utilizzato per risolvere il problema di ottimizzazione. Se la funzione di costo in PO assume valori interi non negativi, e si ha un upper bound U sul costo ottimo, è possibile applicare una **procedura di bisezione**.

Classe P Si indichi con P la **classe dei problemi di decisione per cui esiste un algoritmo di complessità polinomiale**, in grado di risolvere tutte le possibili istanze del problema. Con questo si vuole dire che il *numero di passi*, e quindi la complessità temporale dell'algoritmo, *è limitata superiormente da una funzione polinomiale dello spazio di memoria necessario per descrivere ogni istanza del problema.* Un tale algoritmo è in grado di fare due cose: **generare una soluzione** e **verificarne la correttezza**. Esistono problemi, come K_0, per cui la prima parte del compito è difficile, ma la seconda no. Se si enumerasse, mediante un *albero di ricerca*, tutti i possibili sottoinsiemi J, si potrebbe verificare se una specifica istanza ha risposta positiva o negativa, ma chiaramente tale algoritmo ha **complessità esponenziale**.

Tuttavia, se si disponesse di un ipotetico **calcolatore non deterministico**, in grado di eseguire un numero infinito di processi di calcolo in *parallelo*, si sarebbe in grado di risolvere il problema in tempo polinomiale.

Classe NP Si definisce **classe** NP l'insieme dei **problemi di decisione** le cui istanze che hanno risposta positiva sono *verificabili* in **tempo polinomiale**.

Per definizione, $P \subseteq NP$. Una questione meno ovvia è se valga $P \equiv NP$ o $P \subset NP$ in senso stretto. È ragionevole, da questo punto di vista, cercare di caratterizzare la sottoclasse dei problemi più difficili in NP.

Riduzione in tempo polinomiale Siano P e Q due problemi di decisione, per cui ogni istanza I_P di P può essere trasformata in tempo polinomiale in un'istanza I_Q di Q tale che I_P *ha risposta* ***positiva*** *se e solo se* I_Q *ha risposta positiva.* Si dice che P è **riducibile in tempo polinomiale** a Q ($P \prec Q$.) La notazione $P \prec Q$ sottolinea che *la complessità di* P *non è maggiore della complessità di trasformare* P *in* Q *e poi risolvere* Q,

$$\text{compl}(P) \leq \text{compl}(Q) + \text{compl}(P \to Q).$$

Se la **trasformazione ha una complessità trascurabile**, la riduzione di P a Q mostra che Q non è più facile di P. Se P è difficile e $P \prec Q$, Q non può essere facile. Altrimenti, si potrebbe trasformare un'istanza di P in una di Q, e poi applicare l'algoritmo efficiente per Q.

Problemi NP-difficili Un **problema di decisione** P è detto **NP-difficile** (*NP-hard*) *se ogni problema nella classe* NP *è riducibile a* P. Si indichi con NPH la **classe dei problemi NP-difficili**. (Questa classe è estendibile ai *problemi di ottimizzazione.*)

Problemi NP-completi Un **problema di decisione** P è detto **NP-completo** se è in NP ed è NP-hard. Si indichi con NPC la classe dei **problemi NP-completi**. Le implicazioni di tali definizioni sono

1. Un problema NP-difficile non è più facile di un problema qualsiasi in NP.
2. La classe NPC è la classe dei problemi più difficili in NP.

Per dimostrare che un problema di decisione P è NP-completo, occorre dimostrare che P *è in* NP e *un problema NP-completo* Q *può essere ridotto in tempo polinomiale a* P. Si osserva che, Q è NP-completo, $P \prec Q$, e trascurando la complessità della trasformazione, questo implica $\text{compl}(P) \leq \text{compl}(Q)$. Ma il secondo passo della dimostrazione di NP-completezza, ovvero dimostrare che $Q \prec P$, implica anche che $\text{compl}(Q) \leq \text{compl}(P)$. Le due disuguaglianze, sempre a meno della complessità della trasformazione, mostrano, quindi, che $\text{compl}(Q) = \text{compl}(P)$.

In altre parole, **i problemi della classe NPC sono equivalenti in termini di complessità**

computazionale, e un algoritmo polinomiale per uno di essi permetterebbe di risolverli tutti in tempo polinomiale. Si avrebbe quindi $P = NP$, ipotesi non troppo plausibile a causa dell'equivalenza di una vasta classe di problemi per i quali non è noto un algoritmo di complessità polinomiale, nonostante essi siano stati oggetto di ampio studio nel corso degli anni.

Se si accettasse l'ipotesi $P \neq NP$, è possibile rappresentare le relazioni tra le classi P, NP e NPC come in figura. Essa ipotizza una gerarchia per cui la classe P sarebbe la classe dei problemi più facili in NP, e NPC quella dei problemi più difficili. Tutti i problemi in NP si possono trasformare nel problema $Q \in NPC$. Se $P \in NP$, per dimostrare che $P \in NPC$, occorre trasformare Q in P.

L'innesco della catena Se si dimostrasse che un problema P è in NPC, questo può a sua volta essere trasformato in altri problemi, permettendo di ampliare la classe dei problemi noti in NPC. Il punto critico, chiaramente, è trovare l'innesco della catena, ovvero *il primo problema in NPC, al quale tutti i problemi in NP possono essere ricondotti.*

Il **teorema di Cook** dimostra che il **problema della soddisfacibilità** soddisfa i requisiti necessari e fornisce la soluzione: *data una* ***formula Booleana in forma canonica disgiuntiva****, decidere se esiste un assegnamento di valori ai suoi elementi che la rende vera. Per esempio*, la formula (A or B) and (not(A) or C); definita rispetto alle variabili Booleane A, B e C, è soddisfatta se B e C sono entrambe vere. *Al contrario*, (A or B) and (not(A) or B) and (not(B)) non può essere soddisfatta da alcun assegnamento di verità alle variabili.

A partire dal problema della soddisfacibilità, si può ricavare per riduzioni polinomiali successive una famiglia crescente di problemi NP-completi, compreso il problema K_0, che può essere considerato come un cugino in versione decisionale del problema knapsack.

Teorema Il fatto che tale problema faccia parte della classe NPC permette di dimostrare il teorema seguente, che risolve la questione iniziale. Si consideri una versione decisionale del problema $1/r_i/L_{\max}$: dati i tempi di rilascio r_i, le date di consegna d_i e i tempi di lavorazione p_i, tutti a valori interi positivi, per n job J_i, $i \in [n]$, *esiste una soluzione in cui nessun job è completato in ritardo?* Tale problema di decisione è **NP-completo**.

Dimostrazione Il problema è chiaramente in NP, poiché per una data sequenza è facile verificare se i job vengono completati in tempo rispetto alle due date. Dati gli interi positivi a_j, $j \in [n]$, si creano n job J_j con parametri

$$r_j = 0, \quad p_j = a_j, \quad d_j = 1 + \sum_{k \in [n]} a_k, \qquad j \in [n].$$

Si crei un ulteriore job J_0 con parametri $r_0 = b, \quad p_0 = 1, \quad d_0 = b + 1$. Perché tutti i job rispettino la data di consegna, è necessario che il job J_0 inizi la lavorazione al tempo $t = b$. Inoltre, dato che la data di consegna degli altri job è pari alla somma di tutti i tempi di lavorazione, la soluzione non può presentare periodi di tempo in cui la macchina è ferma, prima di avere completato l'intero insieme di job. Questo richiede che sia possibile individuare un sottoinsieme J di job da schedulare prima di J_0, in modo tale che

$$\sum_{j \in J} p_j = r_0.$$

Tale insieme risolve il problema *subset-sum*.

2.2 Dai problemi di decisione ai problemi di ottimizzazione

Problemi di ottimizzazione Il teorema dimostra che un problema di decisione legato al problema di ottimizzazione $1/r_i/L_{\max}$ è NP-completo, **ma cosa è possibile dire del problema di ottimizzazione stesso?** Per definizione, la classe NPC contiene solo problemi di decisione. É però possibile estendere le classi P e NPH, includendo in esse anche problemi di ottimizzazione.

I **problemi di ottimizzazione** per cui è noto un algoritmo di complessità polinomiale stanno in P. Nella classe NPH è possibile includere problemi di ottimizzazione ai quali si può ricondurre un

corrispondente problema di decisione. **La versione decisionale di $1/r_i/L_{\max}$ si riduce chiaramente al problema di ottimizzazione.**

Inoltre, nella dimostrazione si è assunto che i dati fossero numeri interi. Ma il problema a numeri interi può evidentemente essere ridotto al problema generale. Quindi **il problema di scheduling $1/r_i/L_{\max}$ è NP-difficile**.

L'impatto della codifica di un problema Non è possibile fare a meno di considerare l'**impatto del modo in cui si codifica un problema**. Sarebbe infatti errato, per esempio, associare a un problema *knapsack* una dimensione pari al numero di oggetti. La dimensione si riferisce a una codifica binaria che comprende tutti i dati del problema. Per mostrare la rilevanza di ciò, **si consideri un classico algoritmo di programmazione dinamica** per la soluzione del problema knapsack

$$\begin{aligned} \max \quad & \sum_{k=1}^{n} v_k x_k \\ \text{s.t.} \quad & \sum_{k=1}^{n} w_k x_k \leq B \\ & x_k \in \{0,1\}, \quad k = 1, \ldots, n. \end{aligned}$$

Si definisce la **value function**

$$V_k(s) := \text{valore del sottoinsieme ottimale tra gli oggetti } \{k, k+1, \ldots, n\},$$

quando la capacità residua è s. In sostanza, la value function assume che siano già state fatte scelte di inserimento o meno degli oggetti da 1 a $k-1$; a valle di tale selezione, si ha una capacità residua s, e ci si chiede come utilizzarla al meglio per le scelte rimanenti. Se i dati w_k e B del problema sono interi, lo sarà anche la capacità residua s. Per *risolvere il problema*, ovvero trovare il valore di $V_1(B)$, si applica una relazione ricorsiva

$$V_k(s) = \begin{cases} V_{k+1}(s), & 0 \leq s < w_k, \\ \max\{V_{k+1}(s),\, V_{k+1}(s - w_k) + v_k\}, & w_k \leq s \leq B. \end{cases}$$

Si tratta di una equazione funzionale con condizione terminale

$$V_n(s) = \begin{cases} 0, & 0 \leq s < w_n, \\ v_n, & w_n \leq s \leq B. \end{cases}$$

Occorre tabulare tutte le funzioni $V_k(s)$, $k = 1, \ldots, n$, per valori interi di s, che assume valori nel range da 0 a B. Pertanto, **tale algoritmo ha complessità** $O(nB)$. Questo *non implica che il problema knapsack abbia complessità polinomiale*: per rappresentare il valore B in aritmetica binaria bastano $\log_2 B$ bit. Quindi **l'algoritmo, rispetto a tale codifica binaria, ha complessità esponenziale**. Se si utilizzasse un **computer con una codifica unaria**, l'algoritmo considerato avrebbe **complessità polinomiale**. Si dice infatti che un algoritmo di questo tipo è **pseudo-polinomiale**.

3 CAP 3 – Metodi di decomposizione in ottimizzazione

Recap I **metodi di decomposizione** servono a risolvere problemi **di grande scala** sfruttando **strutture favorevoli**: permettono di **parallelizzare** i calcoli, di gestire modelli di **ottimizzazione stocastica** basati su scenari e di affrontare problemi combinatori difficili tramite una **sequenza di sottoproblemi più semplici**.

La decomposizione nasce quando un problema non è pienamente separabile: in presenza di **fattori complicanti** (vincoli di interazione o variabili di interazione) si ottiene una struttura **block-angular**. Nel caso di **vincoli di interazione** si può decomporre **rilassando** tali vincoli (ad esempio con **decomposizione lagrangiana/duale**); nel caso di **variabili di interazione** si può decomporre **fissando** tali

variabili (logica alla base di **Benders** e della **L-shaped**).

Decomposizione duale e progressive hedging. La **decomposizione duale** usa la **dualità** per rendere separabile un problema: si **dualizza** un vincolo complicante introducendo un **moltiplicatore di Lagrange** interpretabile come **prezzo ombra** della risorsa; così il problema si spezza in sottoproblemi indipendenti e il moltiplicatore viene aggiornato in base allo squilibrio tra consumo e disponibilità (meccanismo tipo **domanda–offerta**). La convergenza può essere lenta e spesso si mira anche a **dual heuristics**. Nella **programmazione stocastica multistadio** con formulazione a variabili per scenario, è cruciale imporre i **vincoli di non anticipatività**: decisioni di scenari indistinguibili fino a un certo tempo devono coincidere. La **progressive hedging** sfrutta una decomposizione **per scenario** e migliora la convergenza combinando **moltiplicatori** e un **termine di penalità quadratica** (interpretato come **regolarizzazione**); la scelta del parametro di penalità e dello schema di aggiornamento dei moltiplicatori è **critica**. Il metodo è particolarmente utile per costruire **euristiche** in problemi **misti-interi** multistadio difficili.

Benders/L-shaped La **decomposizione L-shaped** (equivalente alla **decomposizione di Benders** in programmazione stocastica a due stadi) separa il **primo stadio** dai **sottoproblemi di scenario** del secondo stadio: si risolve un **master** che contiene un'approssimazione della funzione di ricorso e la si raffina con **cutting planes** (tagli di **optimality** e di **feasibility**). In parallelo, l'idea dei **Kelley cutting planes** è costruire progressivamente un **bound inferiore** tramite iperpiani di supporto e iterare finché il gap tra **upper bound** e **lower bound** è sufficientemente piccolo. Varianti come la **regularized L-shaped decomposition** e generalizzazioni multistadio (**Nested Benders**, **Abridged nested Benders**, **Stochastic Dual Dynamic Programming**) mirano a migliorare **convergenza** e scalabilità; nei metodi di **decomposizione stocastica** (**Higle e Sen**) si combina **campionamento** e **ottimizzazione** costruendo **tagli asintoticamente validi**.

Dantzig–Wolfe/column generation. La **decomposizione di Dantzig–Wolfe** e la **generazione di colonne** affrontano modelli con moltissime variabili "utili" ma non enumerabili: si lavora con un **restricted master problem** e si aggiungono nuove colonne risolvendo un **pricing problem** (scelto per individuare colonne con **costo ridotto** favorevole). Nel **cutting stock** la formulazione a **pattern** elimina la **simmetria** del modello diretto, ma richiede generazione dinamica dei pattern; lo schema è alla base di approcci **branch-and-price** e, più in generale, consente di "nascondere" vincoli complessi nel pricing e di risolvere problemi **realmente di grande scala** quando i metodi standard (simplesso o interior point) non sono più competitivi.

Ruolo dei metodi di decomposizione I metodi di **decomposizione** svolgono un ruolo centrale nell'ottimizzazione, poiché consentono di

- Sfruttare **strutture favorevoli** del problema, come: sottoproblemi di rete e trasformazioni da problemi **NP-hard** a problemi risolvibili in **tempo polinomiale**.
- **Parallelizzare** la soluzione di problemi di grande scala.
- Affrontare modelli di **ottimizzazione stocastica** di grandi dimensioni basati su scenari.
- Trattare problemi combinatori difficili mediante una **sequenza di sottoproblemi più semplici**, eventualmente combinando diverse strategie di soluzione.
- Evitare difficoltà di modellazione dovute a **vincoli molto complessi**.

Problema separabile Si consideri il problema di ottimizzazione

$$\begin{aligned} \text{opt} \quad & \sum_{j\in[n]} f_j(x_j) \\ \text{s.t.} \quad & x_j \in S_j. \end{aligned}$$

In questo caso il problema può essere **decomposto** in sottoproblemi indipendenti $\text{opt}_{x_j\in S_j} f_j(x_j)$, poiché i sottovettori x_j sono soggetti a **vincoli indipendenti**.

Nel caso di un **modello di programmazione lineare**

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

la separabilità corrisponde a una struttura **a blocchi diagonali** della matrice A. Nella maggior parte dei casi reali, non si dispone di una struttura così favorevole e sono presenti **fattori complicanti**.

Struttura block-angular Un caso tipico è quello di un **problema LP di grande scala** con struttura **block-angular** della matrice A:

$$A = \begin{bmatrix} C_1 & C_2 & C_3 & \cdots & C_n \\ D_1 & & & & \\ & D_2 & & & \\ & & D_3 & & \\ & & & \ddots & \\ & & & & D_n \end{bmatrix} \quad \text{o} \quad A = \begin{bmatrix} C_1 & D_1 & & & \\ C_2 & & D_2 & & \\ C_3 & & & D_3 & \\ \vdots & & & & \ddots \\ C_n & & & & D_n \end{bmatrix}.$$

Nel primo caso il fattore complicante è rappresentato da un insieme di **vincoli di interazione** che accoppiano i sottoproblemi e impediscono la decomposizione basata sulla struttura a blocchi diagonali. La decomposizione diventa possibile se i vincoli di interazione vengono **rilassati** in qualche modo, ad esempio tramite la **decomposizione lagrangiana duale**.

Nel secondo caso il fattore complicante è costituito da **variabili di interazione**, che impediscono la decomposizione. La decomposizione è possibile se si **fissano** le variabili di interazione, come avviene nella **decomposizione L-shaped**, che corrisponde alla **decomposizione di Benders** nella programmazione stocastica.

Classificazione dei metodi Esiste un ampio insieme di metodi di decomposizione tra loro correlati. Alcuni sono **esatti**, almeno in linea di principio, mentre altri conducono ad **algoritmi approssimati**: **rilassamento lagrangiano e decomposizione lagrangiana**, **euristiche duali**, **decomposizione di Dantzig–Wolfe**, **generazione di colonne**, **decomposizione gerarchica**, **mateuristiche** (da non confondere con le metaeuristiche), **decomposizione di Benders** per MILP con struttura speciale, **decomposizione L-shaped** nella programmazione stocastica a due stadi con ricorso, **progressive hedging** per la programmazione stocastica multistadio con ricorso e **programmazione dinamica** (decomposizione basata sul tempo).

3.1 Decomposizione duale e progressive hedging per la programmazione stocastica

3.1.1 Decomposizione duale

Problema di partenza Per comprendere come la dualità possa essere utilizzata in un contesto semplice, si consideri il problema

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} f_i(x_i) \\ \text{s.t.} \quad & \sum_{i=1}^{n} g_i(x_i) \leq b, \\ & x_i \in S_i, \quad i = 1, \ldots, n. \end{aligned}$$

Le variabili decisionali x_i, $i = 1, \ldots, n$, possono essere interpretate come **attività** che generano un profitto $f_i(x_i)$ e consumano una quantità di risorsa $g_i(x_i)$. La **funzione obiettivo** rappresenta il **profitto totale**,

mentre il **vincolo** è di **budget** sulla risorsa. La funzione obiettivo è misurata in termini monetari, mentre il parametro b è espresso in unità di risorsa. Se fosse possibile eliminare il vincolo di budget, il problema risulterebbe immediatamente **decomponibile**.

Dualizzazione del vincolo di budget Il **vincolo di budget** viene *dualizzato* introducendo un **moltiplicatore di Lagrange** $\mu \geq 0$ e scrivendo la **funzione lagrangiana**:

$$L(x;\mu) = \sum_{i=1}^{n} f_i(x_i) + \mu \left(b - \sum_{i=1}^{n} g_i(x_i) \right) = \sum_{i=1}^{n} \big[f_i(x_i) - \mu g_i(x_i) \big] + \mu b.$$

La funzione lagrangiana deve essere **massimizzata** rispetto alle variabili primali, ottenendo un insieme di sottoproblemi indipendenti

$$\max_{x_i \in S_i} \big[f_i(x_i) - \mu g_i(x_i) \big] \equiv \pi_i(x_i).$$

Ogni sottoproblema consiste nel *massimizzare il contributo di profitto netto*, dato dal profitto meno il costo della risorsa. Il moltiplicatore μ rappresenta un **prezzo ombra** della risorsa, misurato in unità monetarie per unità di risorsa. In questo caso, il problema duale consiste nel **minimizzare** la funzione duale rispetto a μ. Date le soluzioni rilassate x_i^*, un sottogradiente della funzione duale è

$$\sum_{i=1}^{n} g_i(x_i^*) - b.$$

Questo valore è positivo quando il *budget viene superato*; in tal caso il prezzo della risorsa deve essere aumentato. Invece, il prezzo deve invece essere ridotto quando il budget non viene completamente utilizzato. Il meccanismo risultante può essere interpretato come uno schema di **domanda–offerta**, in cui il prezzo della risorsa viene aggiornato in funzione dello squilibrio tra consumo e disponibilità.

Osservazioni sulla dual decomposition La **decomposizione duale può convergere lentamente** nella pratica, ma può risultare efficace per alcuni problemi di grande scala con struttura particolare. Talvolta è sufficiente ottenere una soluzione di buona qualità: se da una decomposizione duale è possibile ricostruire una soluzione primale ammissibile, si ottiene una **dual heuristic**.

I metodi lagrangiani possono essere integrati con **metodi a funzione di penalità**, ottenendo schemi di **Lagrangiano aumentato**, ad esempio minimizzando

$$f(x) + \sum_{i \in I} \lambda_i h_i(x) + \sigma \sum_{i \in I} h_i^2(x),$$

nel caso di problemi con vincoli di uguaglianza.

3.1.2 Copertura progressiva per la programmazione stocastica multistadio

Rappresentazione dell'incertezza Nella **programmazione stocastica multistadio**, l'incertezza è rappresentata tramite **alberi di scenario**. In una **formulazione compatta del modello**, si definiscono variabili decisionali x_n associate ai nodi n dell'albero. In una **formulazione a variabili separate**, si introducono variabili x_t^s, associate allo scenario s al tempo t. Nel secondo caso è necessario imporre esplicitamente i **vincoli di non anticipatività**: variabili decisionali corrispondenti a scenari diversi allo stesso tempo t devono assumere lo stesso valore se gli scenari sono indistinguibili a quel tempo.

Insiemi di scenari indistinguibili Si denoti con $\mathcal{F}_s^t$ l'insieme degli **scenari indistinguibili** da s fino al tempo t. I vincoli di non anticipatività possono essere scritti come: $x_{it}^s = x_{it}^{s'}, \quad \forall i, t, \ s, s' \in \mathcal{F}_s^t$.

Formulazione del problema multistadio La **copertura progressiva** è una strategia che può essere applicata a problemi di programmazione stocastica multistadio, possibilmente non lineari. Si consideri un sistema dinamico con stato z_t e equazione di stato: $z_{t+1} = G_t(z_t, x_t, \xi_{t+1}), \quad t = 0, 1, \dots, T$, dove x_t è la **variabile di controllo** e ξ_{t+1} è una variabile aleatoria.

Sia $s = (\xi_1^s, \xi_2^s, \dots, \xi_{T+1}^s)$ uno **scenario** con probabilità π_s. Il **problema di scenario individuale** è

$$\begin{aligned} \min \quad & \sum_{t=0}^{T} \gamma^t f_t\big(z_t, x_t, \xi_{t+1}^s\big) + \gamma^{T+1} Q(z_{T+1}) \\ \text{s.t.} \quad & z_{t+1} = G_t\big(z_t, x_t, \xi_{t+1}^s\big), \quad t = 0, 1, \dots, T, \\ & L_t(z_t) \le x_t \le U_t(z_t), \quad t = 0, 1, \dots, T. \end{aligned}$$

Sia $(x_0^s, x_1^s, \dots, x_T^s)$ la soluzione ottima del problema associato allo scenario s. Ci si potrebbe chiedere se sia possibile aggregare le soluzioni individuali e scegliere $\underline{x}_t = \sum_{s \in S} \pi_s x_t^s$. Purtroppo, non vi è alcuna ragione per ritenere che tale scelta conduca a una soluzione ottima, o anche solo ammissibile. Una **buona politica decisionale** dovrebbe sfruttare l'informazione disponibile, ma non può essere anticipativa. Il problema multistadio può essere formulato all'interno di un framework di modellazione a variabili replicate per scenario (*split-variable*), introducendo i vincoli di non anticipatività in una forma opportuna.

Sia $\{s\}_t$ l'insieme degli scenari che, fino all'istante di tempo t, non sono distinguibili dallo scenario s, e sia $\mathbb{P}(\{s\}_t)$ la somma delle loro probabilità. Il **problema di ottimizzazione stocastica**complessivo può essere scritto come

$$\begin{aligned} \min \quad & \sum_{s \in S} \pi_s \left(\sum_{t=0}^{T} \gamma^t f_t\big(z_t^s, x_t^s, \xi_{t+1}^s\big) + \gamma^{T+1} Q\big(z_{T+1}^s\big) \right) \\ \text{s.t.} \quad & z_{t+1}^s = G_t\big(z_t^s, x_t^s, \xi_{t+1}^s\big), \quad \forall s,\ t = 0, 1, \dots, T, \\ & L_t(z_t^s) \le x_t^s \le U_t(z_t^s), \quad \forall s,\ t = 0, 1, \dots, T, \\ & x_t^s = \sum_{s' \in \{s\}_t} \frac{\pi_{s'} x_t^{s'}}{\mathbb{P}(\{s\}_t)}, \qquad \forall s \in S,\ t = 0, 1, \dots, T. \end{aligned}$$

In questo caso si utilizza una *singola aspettativa condizionata* per ciascuno scenario e istante temporale, anziché imporre uguaglianze a coppie tra le decisioni associate agli insiemi di scenari indistinguibili. Introduciamo quindi dei **moltiplicatori di Lagrange** w_t^s associati ai vincoli di non anticipatività. La dualizzazione del problema conduce alla seguente formulazione

$$\min \quad \sum_{s \in S} \pi_s \left(\sum_{t=0}^{T} \gamma^t \Big[f_t\big(z_t^s, x_t^s, \xi_{t+1}^s\big) + w_t^s \big(x_t^s - \underline{x}(\{s\}_t)\big) \Big] + \gamma^{T+1} Q\big(z_{T+1}^s\big) \right),$$

dove l'aspettativa condizionata $\underline{x}(\{s\}_t) \equiv \sum_{s' \in \{s\}_t} \frac{\pi_{s'} x_t^{s'}}{\mathbb{P}(\{s\}_t)}$, può essere interpretata come una proiezione sull'insieme delle politiche non anticipative.

Il problema può essere riordinato e decomposto per scenario. Al fine di migliorare la convergenza, è possibile aggiungere un **termine di penalità quadratica**, ottenendo un metodo di tipo *Lagrangiano aumentato*. Purtroppo, l'introduzione diretta del termine di penalità distruggerebbe la struttura di decomposizione. Tuttavia, è possibile utilizzare la soluzione aggregata per scenario $\underline{x}(\{s\}_t)$ ottenuta all'iterazione precedente, il che conduce (trascurando i termini costanti) al seguente problema per il singolo scenario

$$\min \quad \sum_{t=0}^{T} \gamma^t \left\{ f_t\big(z_t^s, x_t^s, \xi_{t+1}^s\big) + w_t^s x_t^s + \frac{\rho}{2} \big[x_t^s - \underline{x}(\{s\}_t)\big]^2 \right\} + \gamma^{T+1} Q\big(z_{T+1}^s\big).$$

Il *termine quadratico* può essere interpretato come un **termine di regolarizzazione**. La scelta del **parametro di penalità** ρ e dello schema di aggiornamento dei moltiplicatori è critica ai fini della convergenza. Ciononostante, il metodo di *progressive hedging* risulta utile per sfruttare la struttura del problema ed è stato ampiamente utilizzato per costruire euristiche per problemi stocastici multistadio misti-interi di difficile soluzione.

3.2 Decomposizione di Benders e decomposizione L-shaped per la programmazione stocastica

Kelley's cutting planes Si consideri il **problema convesso** $\min_{x\in S} f(x)$, e si supponga che, dato un punto x^k, sia possibile calcolare non solo il valore della funzione, ma anche un **subgradiente** γ_k, che esiste se la funzione è convessa sull'insieme S. In altre parole, è possibile trovare una funzione affine tale che

$$f(x^k) = \alpha_k + \gamma_k^T x^k, \qquad f(x) \geq \alpha_k + \gamma_k^T x \;\; \forall x \in S.$$

La disponibilità di un tale iperpiano di supporto suggerisce la possibilità di approssimare f **dal basso**, tramite l'**inviluppo superiore** degli iperpiani di supporto.

L'algoritmo di **Kelley cutting planes** sfrutta questa idea costruendo e migliorando una funzione di bound inferiore fino a soddisfare un criterio di convergenza. Se S è **poliedrale**, occorre risolvere una sequenza di **LP**.

Algorithm 1 Metodo dei piani di taglio di Kelley

1: Sia $x^1 \in S$ una soluzione ammissibile iniziale; **inizializzare** il contatore delle iterazioni $k \leftarrow 0$, l'upper bound $u_0 = f(x^1)$, il lower bound $l_0 = -1$ e la funzione di bound inferiore $\beta_0(x) = -\infty$.
2: **Incrementare** il contatore $k \leftarrow k+1$. Trovare un subgradiente di f in x^k tale che le condizioni $f(x^k) = \alpha_k + \gamma_k^T x^k, \quad f(x) \geq \alpha_k + \gamma_k^T x \;\; \forall x \in S.$ siano soddisfatte.
3: **Aggiornare** l'upper bound ponendo $u_k = \min\{u_{k-1}, f(x^k)\}$ e aggiornare la funzione di bound inferiore ponendo $\beta_k(x) = \max\{\beta_{k-1}(x), \alpha_k + \gamma_k^T x\}$.
4: **Risolvere** il problema $l_k = \min_{x\in S} \beta_k(x)$ e sia x^{k+1} la soluzione ottima ottenuta.
5: Se $u_k - l_k < \varepsilon$, **arrestare l'algoritmo**: x^{k+1} è un'approssimazione soddisfacente della soluzione ottima; altrimenti, tornare a **[2]**.

Proprietà di convessità dei programmi stocastici Si consideri il **problema SLP a due stadi con ricorso**

$$\begin{aligned} \min \;\; & c^T x + \mathbb{E}_\xi\big[Q(x;\xi)\big] \\ \text{s.t.} \;\; & Ax = b, \quad x \geq 0, \end{aligned}$$

dove

$$Q(x,\xi) \equiv \min_y \left\{ q(\xi)^T y \;\middle|\; W(\xi)y = h(\xi) - T(\xi)x, \;\; y \geq 0 \right\}.$$

Si parla di **fixed recourse** quando W è deterministica, mentre si parla di **random recourse** nel caso generale. Si consideri la **funzione di ricorso** $\mathcal{Q}(x) \equiv \mathbb{E}[Q(x,\xi)]$ nel caso di costo di ricorso deterministico

$$Q(x,\xi) \equiv \min_y \left\{ q^T y \;\middle|\; Wy = h(\xi) - T(\xi)x, \;\; y \geq 0 \right\}.$$

Il **dominio effettivo** di $\mathcal{Q}(x)$ consiste dei vettori x che sono ammissibili per i vincoli di primo stadio e tali che $\mathcal{Q}(x) < +\infty$, cioè per cui il problema al secondo stadio è (quasi sicuramente) ammissibile. Si assume che, esclusa l'inammissibilità del secondo stadio, l'attesa sia sempre definita, cioè che le variabili aleatorie non siano heavy-tailed.

Per dualità LP, $Q(x,\xi) = \max_\pi \left\{ \big[h(\xi) - T(\xi)x\big]^T \pi \;\middle|\; W^T\pi \leq q \right\}$. Si denoti l'**insieme ammissibile del duale** con $\Pi = \{\pi \mid W^T\pi \leq q\}$. Poiché q è deterministico, tale insieme è **non aleatorio**.

Dimostrazione di convessità (caso con regione duale non aleatoria) Siano $x_1, x_2 \in \mathcal{D}$, $\lambda \in [0,1]$ e $x_\lambda = \lambda x_1 + (1-\lambda)x_2$. Allora

$$\mathcal{Q}(x_\lambda) = \int_\Omega Q(x_\lambda,\xi)\, dP = \int_\Omega \max_{\pi\in\Pi} \left\{ \big[h(\xi) - T(\xi)x_\lambda\big]^T \pi \right\} dP$$

$$= \int_\Omega \max_{\pi \in \Pi} \left\{ \lambda \left[h(\xi) - T(\xi)x_1\right]^T \pi + (1-\lambda)\left[h(\xi) - T(\xi)x_2\right]^T \pi \right\} dP$$

$$\leq \lambda \int_\Omega \max_{\pi \in \Pi} \left\{ \left[h(\xi) - T(\xi)x_1\right]^T \pi \right\} dP + (1-\lambda) \int_\Omega \max_{\pi \in \Pi} \left\{ \left[h(\xi) - T(\xi)x_2\right]^T \pi \right\} dP$$

$$= \lambda \mathcal{Q}(x_1) + (1-\lambda)\mathcal{Q}(x_2).$$

Convessità della funzione di ricorso La dimostrazione precedente è semplice perché la regione ammissibile duale è non aleatoria. In generale, la convessità della funzione di ricorso può essere mostrata anche in contesti più generali. In sintesi

- La funzione di ricorso è tipicamente **differenziabile** per distribuzioni continue.
- La funzione di ricorso è **poliedrale** per distribuzioni discrete.

In entrambi i casi, si può fare affidamento sui **cutting planes di Kelley** per risolvere il problema. Dato lo **schema di decomposizione** del problema, i cutting planes possono essere ottenuti tramite una strategia di decomposizione.

3.2.1 Decomposizione L-shaped

Modello Si consideri l'equivalente deterministico

$$\begin{aligned} \min \quad & c^T x + \sum_{s \in S} p_s \, q^T y^s \\ \text{s.t.} \quad & Ax = b, \\ & W y^s + T_s x = h_s, \quad \forall s \in S, \\ & x,\ y^s \geq 0. \end{aligned}$$

La matrice tecnologica del problema complessivo ha una struttura **dual block-angular**:

$$\begin{bmatrix} A & 0 & 0 & \cdots & 0 \\ T_1 & W & 0 & \cdots & 0 \\ T_2 & 0 & W & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ T_S & 0 & 0 & \cdots & W \end{bmatrix}.$$

Dato un primo stadio fissato, si ottengono problemi di secondo stadio **indipendenti per scenario**. Si riscrive il **problema a due stadi LP** come

$$\begin{aligned} \min \quad & c^T x + \theta \\ \text{s.t.} \quad & Ax = b, \\ & \theta \geq \mathcal{Q}(x), \\ & x \geq 0. \end{aligned}$$

Si costruisce un **master problem rilassato** rilassando $\theta \geq \mathcal{Q}(x)$, che viene approssimato tramite cutting planes:

- **Optimality cuts** della forma $\theta \geq \alpha^T x + \beta$;
- **Feasibility cuts** della forma $0 \geq \alpha^T x + \beta$.

I coefficienti di ciascun taglio si ottengono risolvendo i sottoproblemi di scenario per una data decisione di primo stadio.

Optimality cuts Sia $\hat{x}$ la soluzione ottima del master iniziale e si consideri il duale del problema di secondo stadio per lo scenario s:

$$Q_s(\hat{x}) \equiv \max(h_s - T_s\hat{x})^T\pi^s \quad \text{s.t. } W^T\pi^s \leq q_s.$$

Data una soluzione duale ottima $\hat{\pi}^s$, valgono

$$Q_s(\hat{x}) = (h_s - T_s\hat{x})^T\hat{\pi}^s$$

$$Q_s(x) \geq (h_s - T_s x)^T\hat{\pi}^s \quad \forall x.$$

La prima deriva dal fatto che $\hat{\pi}^s$ è ottima per $\hat{x}$, ma non necessariamente per un generico x. Sommandola sugli scenari, si ottiene

$$\mathcal{Q}(x) = \sum_{s\in S} p_s Q_s(x) \geq \sum_{s\in S} p_s(h_s - T_s x)^T\hat{\pi}^s.$$

Quindi si aggiunge al master rilassato il seguente **optimality cut**:

$$\theta \geq \sum_{s\in S} p_s(h_s - T_s x)^T\hat{\pi}^s.$$

Feasibility cuts Se il ricorso non è completo, alcuni sottoproblemi di scenario possono essere inammissibili per una decisione di primo stadio $\hat{x}$. Si può allora sfruttare il duale del sottoproblema di scenario per trovare un **feasibility cut** che elimina $\hat{x}$ da ulteriori considerazioni.

Si osservi che la *regione ammissibile del duale non dipende dalle decisioni di primo stadio*, poiché $\hat{x}$ non compare nei vincoli. **Se il duale è inammissibile**, allora il problema di secondo stadio dello scenario corrispondente sarebbe inammissibile per qualsiasi decisione di primo stadio; questo caso viene escluso e può indicare un errore di modellazione.

Quando il **primale è inammissibile**, il duale è illimitato. Esiste dunque un **raggio estremo** della regione duale lungo il quale la soluzione duale tende a infinito. Il duale del secondo stadio è illimitato se esistono variabili duali π^* tali che $W^T\pi^* \leq 0, \quad (h_s - T_s\hat{x})^T\pi^* > 0$. La fattibilità del primale richiede

$$Wy = h_s - T_s x \;\Rightarrow\; (\pi^*)^T Wy = (\pi^*)^T(h_s - T_s x) \leq 0.$$

Quindi $(\pi^*)^T(h_s - T_s x) \leq 0$ è un *taglio valido che viene aggiunto al master rilassato.*

Schema iterativo L-shaped decomposition alterna

1. La soluzione del **master problem rilassato**, che produce $\hat{\theta}$ e $\hat{x}$.
2. La soluzione dei corrispondenti **sottoproblemi di scenario**.

A ogni iterazione, vengono aggiunti tagli al master. L'**algoritmo si ferma** quando la soluzione ottima del master soddisfa $\hat{\theta} \geq \mathcal{Q}(\hat{x})$. Questa condizione può essere rilassata se è sufficiente una soluzione near-optimal.

Variazioni sul tema Esistono varianti per gestire i tagli e migliorare la convergenza (ad esempio la **regularized L-shaped decomposition**). L'idea può essere generalizzata a problemi multistadio: **Nested Benders decomposition**, **Abridged nested Benders decomposition** e **Stochastic Dual Dynamic Programming**.

Nei metodi di **decomposizione stocastica** (**Higle e Sen**) si ha un'interazione tra **campionamento** e **ottimizzazione**: l'**idea è costruire tagli asintoticamente validi**.

Strategie alternative si basano sul vincolare la **funzione di ricorso** superiormente e inferiormente, **partizionando il dominio** per ottenere **bound stretti**. Quando i **metodi del simplesso** o gli **interior point methods** riescono a risolvere un **LP stocastico**, spesso non c'è vantaggio nell'uso della decomposizione; tuttavia, la **decomposizione** è necessaria per problemi **realmente di grande scala**.

3.3 Decomposizione Dantzig–Wolfe e generazione di colonne

Problema di cutting stock Si devono tagliare rotoli (tutti di larghezza L) di un certo materiale in rotoli più corti, in modo da soddisfare una domanda assegnata. Si introduca un insieme di n item con domanda d_i e larghezza $w_i \leq L$, $i \in [n]$. L'**obiettivo** è impacchettare gli item in modo da utilizzare il **numero minimo di rotoli**. Questo è un esempio stilizzato di una vasta classe di problemi noti come **cutting-stock problems**.

Modello MILP Siano disponibili m rotoli, indicizzati da $j \in [m]$, e si introducano le variabili decisionali:

- $\delta_j \in \{0,1\}$, uguale a 1 se si usa il rotolo j.
- $x_{ij} \in \mathbb{Z}_+$, numero di item i tagliati dal rotolo j.

$$\begin{aligned} \min \quad & \sum_{j\in[m]} \delta_j \\ \text{s.t.} \quad & \sum_{i\in[n]} w_i x_{ij} \leq L\delta_j, \quad j \in [m], \\ & \sum_{j\in[m]} x_{ij} = d_i, \quad i \in [n], \\ & \delta_j \in \{0,1\}, \quad x_{ij} \in \mathbb{Z}_+, \quad i \in [n],\ j \in [m]. \end{aligned}$$

Pur essendo corretto dal punto di vista teorico, il modello soffre di difficoltà legate a vincoli **big-M** e a problemi di **simmetria** (molte soluzioni ottime equivalenti possono penalizzare i solver commerciali).

Modello a pattern di taglio Si passa a una formulazione basata su un insieme di **cutting patterns**. Ogni pattern, indicizzato da k, è associato al numero a_{ik} di item i ottenuti da un rotolo. Dato un insieme K di pattern, si introduce il numero intero γ_k di rotoli tagliati secondo il pattern k

$$\begin{aligned} \min \quad & \sum_{k\in K} \gamma_k \\ \text{s.t.} \quad & \sum_{k\in K} a_{ik}\,\gamma_k = d_i, \quad i \in [n], \\ & \gamma_k \in \mathbb{Z}_+, \quad k \in K. \end{aligned}$$

Questa formulazione risolve la simmetria, ma la cardinalità di K è enorme: solo pochi pattern sono davvero utili e non è ovvio come restringere K *a priori*. Si parte quindi con un numero limitato di pattern e li si genera dinamicamente tramite **column generation**.

Generazione di una nuova colonna (by pricing) Si consideri un LP-rilassato di una versione ristretta del modello

$$\begin{aligned} \min \quad & \sum_{k\in K'} \gamma_k \\ \text{s.t.} \quad & \sum_{k\in K'} a_{ik}\,\gamma_k = d_i, \quad i \in [n], \\ & \gamma_k \geq 0, \quad k \in K'. \end{aligned}$$

dove $K' \subset K$. Come migliorare la soluzione introducendo una nuova colonna?

Siano π_i, $i \in [n]$, le variabili duali del vincolo $\sum_{k\in K'} a_{ik}\,\gamma_k = d_i$. Dalla teoria del simplesso primale, conviene introdurre in base una variabile con **costo ridotto negativo**. Per un pattern alternativo k, il costo ridotto è

$$\hat{c}_k = 1 - \sum_{i\in[n]} \pi_i a_{ik}.$$

Si cerca la colonna q con costo ridotto minimo, $\hat{c}_q = \min_k \hat{c}_k$. Se $\hat{c}_q \geq 0$, la base corrente è ottima; altrimenti si introduce la variabile non basica γ_q e si itera.

Pricing come knapsack La minimizzazione si ottiene risolvendo un **problema intero di knapsack**

$$\begin{aligned} \max \quad & \sum_{i\in[n]} \pi_i y_i \\ \text{s.t.} \quad & \sum_{i\in[n]} w_i y_i \le L, \\ & y_i \in \mathbb{Z}_+. \end{aligned}$$

Questo produce un nuovo pattern, che viene introdotto se il valore ottimo è maggiore di 1. Ciò fornisce un **lower bound** per un algoritmo **branch-and-price**.

In alternativa, si può generare un insieme di colonne e poi ripristinare l'integralità delle variabili γ_k risolvendo un singolo IP (euristica). In casi meno stilizzati, la complessità dei vincoli può essere "nascosta" nel sottoproblema di pricing. Lo schema di column generation è al cuore della **decomposizione di Dantzig–Wolfe**.

3.3.1 Decomposizione di Dantzig–Wolfe

Modello LP Si consideri un modello LP

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \ge b, \\ & Dx \ge f, \\ & x \ge 0_n. \end{aligned}$$

Si separano i vincoli interpretando $Ax \ge b$ come vincoli **complicanti**, mentre $Dx \ge f$ (insieme alla non-negatività) come vincoli **facili**.

Si definisce il **poliedro** $X := \{x \in \mathbb{R}^n \mid Dx \ge f,\ x \ge 0_n\}$. Si assume che minimizzare una funzione lineare su X sia facile (ad esempio perché D è block-diagonal o per struttura favorevole come un network flow). Per il teorema di **Minkowski–Weyl**, X si può rappresentare come somma di un politopo e di un cono: $X = \text{conv}\{V\} + \text{cone}\{D\}$, dove $V = \{v^1, \dots, v^Q\}$ è l'insieme dei punti estremi e $D = \{d^1, \dots, d^R\}$ l'insieme delle direzioni estreme.

Rappresentazione mediante combinazioni convesse e coniche Il vincolo $x \in X$ si esprime come

$$\begin{aligned} & x = \sum_{q\in V} \lambda_q v^q + \sum_{r\in D} \mu_r d^r \\ & \sum_{q\in V} \lambda_q = 1, \\ & \lambda_q \ge 0, \quad q \in V; \qquad \mu_r \ge 0, \quad r \in D. \end{aligned}$$

dove il secondo vincolo è **di convessità**. Sostituendo il valore della x nel modello originale si ottiene un problema in λ_q e μ_r.

Definendo gli scalari e vettori $c_q := c^T v^q, \quad a_q := Av^q, \quad q \in V, \qquad c_r := c^T d^r, \quad a_r := Ad^r, \quad r \in D,$ si ottiene il **master problem** (MP)

$$\begin{aligned} \min \quad & \sum_{q\in V} c_q \lambda_q + \sum_{r\in D} c_r \mu_r \\ \text{s.t.} \quad & \sum_{q\in V} a_q \lambda_q + \sum_{r\in D} a_r \mu_r \ge b, \\ & \sum_{q\in V} \lambda_q = 1, \\ & \lambda_q \ge 0, \quad q \in V; \qquad \mu_r \ge 0, \quad r \in D. \end{aligned}$$

Il problema è che *MP può coinvolgere un numero enorme di variabili* e non è in generale possibile enumerare tutti i punti/direzioni estremi di X a partire dalla descrizione per disuguaglianze.

Restricted Master Problem (RMP) La strategia si basa su **column generation**. Si inizializza un **Restricted Master Problem** con un sottoinsieme $V' \subset V$ di punti estremi e $D' \subset D$ di direzioni estreme:

$$\begin{aligned} \min \quad & \sum_{q \in V'} c_q \lambda_q + \sum_{r \in D'} c_r \mu_r \\ \text{s.t.} \quad & \sum_{q \in V'} a_q \lambda_q + \sum_{r \in D'} a_r \mu_r \geq b, \\ & \sum_{q \in V'} \lambda_q = 1, \\ & \lambda_q \geq 0, \quad q \in V'; \qquad \mu_r \geq 0, \quad r \in D'. \end{aligned}$$

Siano π e π^0 le variabili duali associate rispettivamente ai vincoli.

Costo ridotto per punti estremi e direzioni estreme Nel simplesso standard, una colonna entra in base se il costo ridotto è negativo. Per i punti estremi, si verifica se esiste $q \in V$ tale che

$$\bar{c_q} = c_q - \begin{bmatrix} \pi^T & \pi^0 \end{bmatrix} \begin{bmatrix} a_q \\ 1 \end{bmatrix} = c_q - \pi^T a_q - \pi^0 = c_q - \pi^T A v^q - \pi^0 < 0.$$

Per le direzioni estreme:

$$\bar{c_r} = c_r - \begin{bmatrix} \pi^T & \pi^0 \end{bmatrix} \begin{bmatrix} a_r \\ 0 \end{bmatrix} = c_r - \pi^T a_r = c_r - \pi^T A d^r < 0.$$

In linea di principio si dovrebbe risolvere il **problema ausiliario**

$$\min \left\{ \min_{q \in V} \bar{c_q}, \ \min_{r \in D} \bar{c_r} \right\},$$

ma non è affrontabile per enumerazione completa. Si riconduce allora a un **pricing problem**

$$\begin{aligned} \min \quad & (c^T - \pi^T A)\, x \\ \text{s.t.} \quad & Dx \geq d, \\ & x \geq 0_n. \end{aligned}$$

Si trascura il termine costante π^0 e si usa lo stesso problema per trovare un vettore $x \in \mathbb{R}^n$ che giochi il ruolo di punto estremo o direzione estrema. Sia z^*_{PP} il valore ottimo del problema di pricing

- Se $z^*_{PP} = -\infty$ (PP illimitato inferiormente), si è trovata una **direzione estrema** x^* con costo ridotto negativo; si aggiunge a RMP una nuova variabile μ_r associata alla colonna
$$\begin{bmatrix} Ax^* \\ 0 \end{bmatrix}.$$
- Se $-\infty < z^*_{PP} < \pi^0$ (PP limitato e valore inferiore alla variabile duale del vincolo di convessità), si è trovato un **nuovo punto estremo** x^* con costo ridotto negativo; si aggiunge a RMP una nuova variabile λ_q associata alla colonna
$$\begin{bmatrix} Ax^* \\ 1 \end{bmatrix}.$$
- Se $z^*_{PP} \geq \pi^0$, non esiste alcun punto/direzione utile da aggiungere e ci si può fermare.

In termini intuitivi, le variabili duali ottime ottenute risolvendo RMP suggeriscono una direzione $(c - A^T\pi)$ lungo la quale si "sonda" il poliedro X. Se X è un politopo si può trovare solo un **punto estremo**; se X è illimitato si può trovare una **direzione estrema**. Non serve trovare tutti i punti e tutte le direzioni: interessano solo quelli con costo ridotto negativo, perché possono migliorare il valore ottenuto risolvendo RMP.

4 CAP 4 – Sistemi MRP/ERP e approccio JIT

Recap I metodi classici di gestione delle scorte risultano inadeguati in presenza di strutture di prodotto complesse, vincoli di capacità e contesti non *make-to-stock*, poiché la propagazione dei fabbisogni lungo la distinta base può generare una marcata **amplificazione della variabilità**. In tale contesto si colloca l'evoluzione dalla **logica MRP** ai sistemi **MRPII** ed **ERP**, come risposta operativa al problema del *lot-sizing multilivello*.

La **logica MRP** si fonda sull'assunzione di **capacità infinita**, in cui il vincolo di capacità è surrogato tramite un **lead time fissato a priori**, e su una procedura ricorsiva di esplosione dei fabbisogni a partire dal **Master Production Schedule**. Gli ordini pianificati non sono esecutivi e il processo è intrinsecamente soggetto al fenomeno del **nervosismo**, per cui piccole variazioni dell'MPS possono produrre grandi variazioni negli ordini, anche per effetto di bordo legato alla ripianificazione a *rolling horizon*.

A livello di *shop floor*, la **Factory Physics** fornisce una chiave di lettura fondamentale delle prestazioni dei sistemi produttivi, mettendo in relazione **throughput**, **flow time** e **work in process** attraverso la **legge di Little**. Tale relazione evidenzia come, a parità di throughput, la riduzione del WIP sia l'unico modo strutturale per ridurre i tempi di attraversamento, e come un'elevata utilizzazione del sistema, in presenza di variabilità, conduca a un'esplosione dei tempi di attesa e del WIP. La variabilità, sia prevedibile sia imprevedibile, rappresenta quindi un fattore chiave nel deterioramento delle prestazioni operative e spiega il ricorso sistematico, nei sistemi tradizionali, a buffer sotto forma di scorte, capacità in eccesso o lead time gonfiati.

In contrapposizione alla logica **push** tipica dell'MRP, l'approccio **Just-In-Time (Toyota)** affronta il problema alla radice, mirando a **ridurre la variabilità alla fonte** piuttosto che ad assorbirla a valle. Ciò avviene attraverso la **produzione livellata** (*production smoothing*), la riduzione sistematica dei **tempi di setup**, il controllo **pull** del WIP e l'adozione di flussi regolari e ripetitivi. Strumenti come il **sistema kanban** e il **goal chasing** consentono di sincronizzare il consumo dei componenti con l'avanzamento della linea di assemblaggio, rendendo possibile il controllo delle linee a monte tramite segnali fisici di prelievo. L'approccio Toyota mette così in luce il legame strutturale tra variabilità, livelli di magazzino e prestazioni del sistema produttivo, mostrando come la riduzione del WIP e dei lead time sia inseparabile da una gestione coerente della variabilità e dell'utilizzazione del sistema.

4.1 Limiti degli approcci classici di gestione delle scorte

Inadeguatezza dei modelli tradizionali I classici approcci di controllo delle scorte presentano **limiti severi** nei seguenti contesti

- Ambienti *non make-to-stock*, come *make-to-order* e *assemble-to-order*, in cui viene ignorata la variabilità prevedibile.
- Presenza di **vincoli di capacità produttiva**, per cui tali modelli risultano più adatti a problemi retail.
- Strutture di prodotto **complesse**, rappresentate mediante distinte base multilivello.

Effetto di amplificazione della variabilità Anche in presenza di una domanda regolare per il prodotto finito, la propagazione dei fabbisogni lungo la distinta base può indurre un'**amplificazione della variabilità** sui componenti a valle.

4.2 Evoluzione dai modelli MRP ai sistemi ERP

Lot-sizing multilivello e difficoltà computazionali In linea di principio è possibile costruire un *modello MILP* per il problema di **lot-sizing multilivello**, collegando domanda indipendente e domanda dipendente. Tali modelli risultano tuttavia difficili da risolvere, e erano computazionalmente impraticabili fino agli anni Settanta.

Negli anni Settanta sono stati introdotti i sistemi **MRP (Material Requirements Planning)**, basati su una **logica a capacità infinita**. Il vincolo di capacità produttiva viene rilassato, consentendo un **disaccoppiamento parziale tra item**, ma non tra domanda indipendente e dipendente. L'interazione tra item viene anticipata e surrogata mediante un **lead time fissato a priori**.

Evoluzione verso MRPII ed ERP L'evoluzione successiva ha portato ai sistemi **MRPII (Manufacturing Resource Planning)**, che includono strumenti di verifica della capacità come **RCCP** (*Rough Cut Capacity Planning*) e **CRP** (*Capacity Requirement Planning*): per verificare il soddisfacimento dei vincoli di capacità. L'ulteriore evoluzione è rappresentata dai sistemi **ERP (Enterprise Resource Planning)**, caratterizzati dall'integrazione con le funzioni commerciali e finanziarie.

4.3 Logica MRP

Assunzione di capacità infinita La logica MRP assume **capacità infinita**: il vincolo di capacità produttiva non viene considerato esplicitamente, ma è rappresentato indirettamente dal lead time.

Il **lead time** impone di lanciare gli ordini di produzione o di acquisto con sufficiente anticipo rispetto alla domanda. Il meccanismo di **lead time offsetting** consente di anticipare gli ordini pianificati rispetto ai fabbisogni.

Record MRP Ogni codice è descritto mediante un record MRP articolato per periodo, contenente: fabbisogni lordi, consegne attese, magazzino disponibile, fabbisogni netti e ordini pianificati.

Prima del lead time offsetting è necessario calcolare i **fabbisogni netti**, ottenuti nettificando i fabbisogni lordi tenendo conto di: giacenze disponibili (*on-hand*) e ordini già emessi (*on-order*).

4.4 Esplosione dei fabbisogni e struttura del processo MRP

Domanda indipendente e MPS Per i prodotti finiti (*end item*), che costituiscono la radice della distinta base, i fabbisogni lordi sono definiti dal **Master Production Schedule (MPS)**. La logica MRP procede in modo **ricorsivo** a partire dagli end item, scendendo lungo le distinte base ed esplodendo i fabbisogni di ciascun codice fornendo i tempi di lancio degli ordini a tutti i livelli.

Ordini pianificati, rilascio e regole di lot-sizing I *lotti* corrispondono ai fabbisogni netti secondo la regola **lot-for-lot**. È importante comprendere la dinamica di un sistema MRP: gli ordini pianificati **non sono ordini esecutivi**; gli ordini pianificati dell'*action bucket*, una volta rilasciati, vengono trasformati in consegne attese. Un'ulteriore differenza tra ordini pianificati e ordini operativi è che, quando un ordine di produzione viene rilasciato, vengono modificati i record relativi alle giacenze di magazzino dei componenti; una parte della giacenza viene allocata per l'ordine ed è da considerarsi non disponibile nel calcolo dei fabbisogni netti.

I **pacchetti MRP** permettono di specificare un orizzonte temporale di *release*; solo gli ordini che ricadono all'interno di tale orizzonte dovrebbero essere rilasciati, in quanto gli altri sono soggetti a incertezze eccessive. Nel calcolo dei fabbisogni è possibile tenere conto di scorte di sicurezza; in questo caso, i fabbisogni netti vengono generati non quando il magazzino disponibile va sotto zero, ma *quando va sotto un certo livello di soglia*. Esiste una gamma di regole di lot-sizing, a quantità fissa o variabile, oppure euristiche per la minimizzazione dei costi totali di giacenza e di ordinazione.

4.5 Il problema del nervosismo

Nervosismo Il dimensionamento dei lotti a partire dai livelli alti della distinta base può generare effetti non intuitivi in presenza di variazioni del MPS. Le regole a quantità variabile sono soggette al fenomeno del **nervosismo** (*nervousness*).

Il nervosismo si manifesta quando **piccole variazioni nei fabbisogni** producono **grandi variazioni**

negli ordini pianificati, anche a livelli inferiori della distinta base. Il fenomeno può portare a: ordini urgenti, instabilità del piano e risultati anti-intuitivi.

Effetto di bordo, nervosismo e strategie di mitigazione Un altro punto da considerare è l'**effetto di bordo** dovuto alla ripianificazione *rolling horizon*, in cui si aggiunge un *time bucket* all'orizzonte di pianificazione. L'aggiunta del fabbisogno relativo a tale time bucket può alterare l'accorpamento dei fabbisogni, con esiti imprevedibili.

Esistono diversi modi per evitare il fenomeno del nervosismo. L'adozione di regole a quantità fissa permette di filtrare naturalmente variazioni di piccola entità, al costo di un aumento nel livello delle scorte. È anche possibile adottare strategie di gestione differenziata dell'orizzonte temporale di pianificazione, dette strategie di **time fencing**. Nell'immediato non è possibile cambiare nulla nell'MPS; in un periodo intermedio è possibile alterare l'MPS solo dopo specifica analisi e autorizzazione; è invece possibile cambiare l'MPS a piacere nei periodi più lontani.

Inoltre, uno strumento utile per evitare il nervosismo e per risolvere situazioni anomale è l'uso dei **firm planned orders**; si tratta di ordini che non possono essere modificati dall'MRP quando le condizioni cambiano, ma soltanto dietro istruzione del pianificatore.

4.6 Master Production Scheduling e Legge di Little

Master Production Scheduling (MPS) Il **Master Production Schedule (MPS)** costituisce l'input primario per la logica MRP ed è basato in parte su ordini cliente e in parte su attività di *forecasting* (demand planning). Nella logica MRP tradizionale, l'MPS può essere validato mediante moduli **RCCP (Rough Cut Capacity Planning)**. Non è detto che l'MPS venga costruito esclusivamente per i codici alla radice delle distinte base: può esistere domanda indipendente per parti di ricambio e, in contesti *assemble-to-order* (ATO), può essere preferibile adottare una struttura a due livelli, con un MPS a livello di moduli e un *Final Assembly Scheduling* a livello superiore.

Capacity Requirements Planning (CRP) La logica MRP è basata su un'assunzione di **capacità infinita**. È tuttavia possibile effettuare una verifica a posteriori del carico di lavoro rispetto alla capacità effettivamente disponibile tramite moduli di **Capacity Requirements Planning (CRP)**. La risoluzione manuale di eventuali situazioni di non ammissibilità risulta complessa, a meno di ricorrere a modelli di ottimizzazione o a procedure euristiche a capacità finita. Inoltre, per evitare ritardi, si tende spesso a gonfiare il lead time presunto, generando work in process che a sua volta allunga ulteriormente i lead time, dando luogo a un potenziale **circolo vizioso**.

Factory Physics A livello di *shop floor*, le misure di prestazione fondamentali sono il **throughput**, il **flow time** (strettamente legato al lead time e comprensivo di tempi di attesa, lavorazione e movimentazione) e il **work in process (WIP)**, associato ai materiali in coda. Idealmente, si desidererebbe un throughput elevato con flow time e WIP contenuti; tuttavia, il raggiungimento simultaneo di tali obiettivi è fortemente influenzato dalla variabilità, sia prevedibile sia imprevedibile.

Legge di Little Un risultato fondamentale della teoria delle code è la **legge di Little**, che esprime la relazione tra WIP, throughput e flow time

$$\text{WIP} = \text{throughput} \times \text{flow time}.$$

La legge mette in evidenza il legame strutturale tra il livello di materiali in lavorazione e il tempo di attraversamento del sistema.

Modello di una singola macchina Si consideri una singola macchina dotata di un buffer per il WIP. Si introducono le seguenti grandezze: il tempo medio di attesa in coda W_q, il tempo medio di lavorazione t_s, il tasso medio di servizio $\mu = 1/t_s$, il tasso medio di arrivo λ, che in condizioni di equilibrio coincide

con il throughput, e la lunghezza media della coda L, che rappresenta il WIP. In tale contesto, la legge di Little può essere riscritta come

$$L = \lambda\,(W_q + t_s).$$

L'**utilizzazione del sistema** è definita come $u = \lambda/\mu$, ed è compresa tra 0 e 1.

Effetto della variabilità Una **coda** del tipo $G/G/1$ non è trattabile analiticamente in forma chiusa; tuttavia, una formula approssimata, esatta nel caso $M/M/1$, fornisce una stima del tempo medio di attesa in coda

$$W_q \approx \left(\frac{C_a^2 + C_s^2}{2}\right)\left(\frac{u}{1-u}\right)t_s,$$

dove C_a e C_s sono i **coefficienti di variazione dei tempi di interarrivo e di servizio**, rispettivamente. La relazione mostra come l'attesa in coda cresca rapidamente all'aumentare dell'utilizzazione e della variabilità.

Buffering Law Per ovviare agli effetti della variabilità è necessario introdurre dei **buffer**, che possono assumere la forma di magazzino o WIP, capacità in eccesso oppure tempo, sotto forma di lead time gonfiato. In assenza di una riduzione della variabilità, il sistema paga un prezzo in termini di WIP elevato, capacità sottoutilizzata e peggioramento del livello di servizio al cliente, manifestato da vendite perse, lead time lunghi e consegne in ritardo.

Produzione livellata e fonti di variabilità Uno dei fondamenti dell'approccio *Toyota* tradizionale è la **produzione livellata** (*production smoothing*). La variabilità non è legata esclusivamente a eventi casuali, come i guasti alle macchine, ma può derivare anche da batching dovuto ai tempi di setup, batching nella movimentazione (*wait to move*), scarso coordinamento nell'assemblaggio (*wait to match*) e variabilità della domanda riflessa nell'MPS.

4.7 Approccio Just-In-Time (Toyota)

Principi dell'approccio Just-In-Time L'approccio **Just-In-Time (JIT)**, sviluppato nell'ambito del sistema Toyota, si basa sull'idea di ridurre la variabilità mediante la ripetizione di un **mix di produzione ripetitivo** (*mixed-model*), realizzando una produzione livellata (*production smoothing*).

Il presupposto fondamentale è la *riduzione o l'annullamento dei tempi di setup*, che consente di produrre lotti piccoli e frequenti. Il controllo del **work in process** avviene tramite un approccio **pull**, basato sul prelievo fisico dei materiali, in contrapposizione alla logica **push**, basata sul rilascio di ordini pianificati a partire da previsioni di fabbisogno. Lo strumento chiave del controllo pull è il **sistema kanban**, con possibile alternativa nel sistema **CONWIP**.

Sequenziamento e regolarità dei flussi A parità di mix produttivo, esistono diverse sequenze di assemblaggio che realizzano lo stesso numero di prodotti finiti. Il problema del **Toyota Goal Chasing** consiste nello scegliere una sequenza che renda il più possibile **regolare il fabbisogno di componenti** sulle linee laterali che alimentano la linea principale di assemblaggio. *L'obiettivo è consentire il controllo delle linee a monte mediante un sistema pull, che richiede flussi regolari e prevedibili.*

Formalizzazione del problema Si consideri una linea di assemblaggio che realizza N prodotti finiti sulla base di M moduli prodotti su linee laterali, con una distinta base piatta. Sia b_{ij} il numero di componenti di tipo j richiesti per assemblare un'unità del prodotto finito i, e sia Q_i il numero di finiti di tipo i previsti nel mix ripetitivo. Il fabbisogno per ciclo dei componenti di tipo j è dato da

$$N_j = \sum_{i=1}^{N} b_{ij} Q_i.$$

Indicando con $Q = \sum_{i=1}^{N} Q_i$ il numero totale di assemblaggi per ciclo, il **consumo cumulato ideale** dei componenti di tipo j al passo k dovrebbe essere pari a $\frac{kN_j}{Q}$.

Funzione obiettivo del goal chasing Sia X_{jk} il *consumo cumulato effettivo* del componente j al passo k. L'obiettivo del problema di goal chasing è **minimizzare la distanza tra consumo ideale e consumo reale**, misurata tramite la funzione

$$\sum_{k=1}^{Q} \sum_{j=1}^{M} \left(\frac{kN_j}{Q} - X_{jk} \right)^2 .$$

Ruolo dei tempi di setup nel JIT Nel JIT viene posta forte enfasi sulla riduzione dei **tempi di setup**, poiché essa consente di ridurre la dimensione dei lotti e, di conseguenza, il livello medio delle giacenze. Tuttavia, l'impatto dei tempi di setup deve essere analizzato congiuntamente agli altri fattori del sistema produttivo. In particolare, la riduzione dei livelli di magazzino non dipende esclusivamente dalla diminuzione dei setup, ma anche dal tasso di produzione e dal grado di saturazione del sistema.

4.7.1 Modello di rotazione ciclica dei prodotti

Impostazione del modello Si consideri una linea su cui ruotano ciclicamente N prodotti. Per ciascun prodotto i si introducono il tasso di produzione p_i (pezzi per unità di tempo), il tasso di domanda $d_i < p_i$, assunto costante, e il tempo di setup s_i, assunto indipendente dalla sequenza. Il periodo di rotazione T_c deve essere ridotto al fine di contenere il livello medio di giacenza.

Indicando con T_i la durata del lotto del prodotto i in ciascuna rotazione, in condizioni di equilibrio deve valere

$$p_i T_i = d_i T_c \qquad \Rightarrow \qquad T_i = \frac{d_i}{p_i} T_c.$$

Inoltre, il **periodo di rotazione** deve soddisfare il vincolo

$$T_c \geq \sum_{i=1}^{N} s_i + \sum_{i=1}^{N} T_i.$$

L'eventuale *slack* è utile per assorbire guasti, ritardi ed effettuare attività di manutenzione.

Limite inferiore del periodo di rotazione Sostituendo l'espressione di T_i nel vincolo precedente, si ottiene il **limite inferiore**

$$T_c \geq \frac{\sum_{i=1}^{N} s_i}{1 - \sum_{i=1}^{N} \frac{d_i}{p_i}}.$$

Il risultato mostra che, oltre ai tempi di setup, il periodo di rotazione è fortemente influenzato dal *rapporto tra tassi di domanda e tassi di produzione*, evidenziando un legame diretto con i fenomeni di congestione già osservati nei modelli di coda.

5 CAP 5 – Schedulazione nella produzione e nei servizi

Recap La **schedulazione** affronta l'assegnazione di **risorse** a **job** nel tempo, rispettando **vincoli tecnologici** e di **capacità** e ottimizzando una **misura di prestazione**. Rispetto alla **pianificazione della produzione** (ad esempio a livello **MRP**), la schedulazione opera a un livello di **dettaglio superiore**, assumendo come dati tempi di lavorazione, precedenze e vincoli operativi.

I job sono descritti come sequenze di **operazioni** soggette a **precedenze**, **tempi di processo**, **date di rilascio** e **due date**. Le assunzioni classiche prevedono lavorazione su **una macchina per volta**, **una sola operazione per macchina** e **assenza di interruzioni**, pur ammettendo varianti come **preemption**, **batch** e **lot streaming**. Le diverse strutture di flusso (**macchina singola**, **parallele**, **flow shop**, **job shop**, **open shop**) determinano la complessità del problema. Le prestazioni sono valutate

tramite **funzioni di penalità** dei tempi di completamento, come C_i, F_i, L_i, T_i, E_i e U_i, eventualmente aggregate in forma **min-sum** o **min-max** e pesate tramite **priorità** w_i. Alcune misure risultano **equivalenti** fino a costanti, mentre $\mathbf{L}_{\max}$ implica l'ottimalità anche per $\mathbf{T}_{\max}$.

Nel caso di **misure di prestazione regolari**, ossia **non decrescenti** nei tempi di completamento, il **timing** può essere ricavato direttamente dal **sequencing** e la ricerca può essere limitata a **soluzioni attive**. Per **misure non regolari** (come earliness–tardiness), sequencing e timing non coincidono e la soluzione ottima non è necessariamente **attiva né semiattiva**, rendendo il problema più complesso.

La **notazione di Graham** $\alpha|\beta|\gamma$ fornisce una classificazione compatta dei problemi di schedulazione in base al layout delle macchine, ai vincoli aggiuntivi e alla misura obiettivo. Alcuni casi ammettono soluzioni **polinomiali** (regole **EDD**, **WSPT**, algoritmo di **Johnson**), ma nella maggior parte dei casi i problemi sono **NP-hard** e richiedono **euristiche** e **metaeuristiche**. La modellazione tramite **grafo disgiuntivo** rappresenta vincoli tecnologici e di capacità, e il **cammino critico** determina il **makespan**. Da tale rappresentazione deriva un modello **MILP** per $J//C_{\max}$, utile come base concettuale per strategie di ricerca locale. In questo contesto, solo perturbazioni degli archi disgiuntivi sul **cammino critico** sono efficaci.

La procedura **shifting bottleneck** decompone il problema $J//C_{\max}$ in una sequenza di problemi su macchina singola del tipo $1/r_i/L_{\max}$, utilizzando **teste** e **code** per costruire **due date locali**. Il collo di bottiglia viene identificato iterativamente come la macchina con $L_{\max}$ peggiore e congelato fino alla completa schedulazione del sistema.

5.1 Modelli classici di machine scheduling

Scheduling Risolvere un problema di **schedulazione** richiede di assegnare **risorse** a **job** da eseguire nel tempo, rispettando **vincoli tecnologici** e di **capacità**, in modo da ottimizzare una **misura di prestazione**. Un **problema di scheduling** si pone a un livello di dettaglio superiore rispetto a un problema di **pianificazione della produzione** (ad esempio a livello **MRP**). Esiste una grande varietà di problemi di scheduling, ma si inizia con la definizione di un **problema minimale**.

Caratteristiche dei job I job sono caratterizzati da

- Un **set di operazioni** da eseguire, legate da **vincoli di precedenza**.
- **Tempi di lavorazione** per l'esecuzione su ogni macchina.
- **Date di rilascio** e **date di consegna**.

Assunzioni comuni sono che un job possa essere in lavorazione su **al più una macchina per volta**, che ogni macchina possa lavorare **un job per volta**, e che le lavorazioni **non possano essere interrotte**. Ognuna di tali assunzioni può essere violata (**lot streaming**, **processori batch**, **pre-emption**).

Soluzioni e diagrammi di Gantt Una soluzione di un problema di scheduling è caratterizzata dalle **sequenze di lavorazione** sulle macchine. Una soluzione può essere visualizzata mediante un **diagramma di Gantt**, che rappresenta graficamente l'allocazione temporale dei job sulle macchine.

5.2 Tipi di flusso

Strutture di precedenza Esistono diverse **strutture di precedenza** tra le operazioni dei job, che caratterizzano la **tecnologia del ciclo di lavorazione**. Sono possibili **strutture lineari**, oppure **strutture arbitrarie** rappresentabili mediante **grafi**. Le precedenze si traducono in **vincoli** di un problema di ottimizzazione. Il **flusso dei materiali** caratterizza diverse strutture in termini di macchine

- **Macchina singola**.
- **Macchine parallele** (identiche: $p_{im} = p_i$, correlate: $p_{im} = p_{ism}$, scorrelate).

- **Flow shop**.
- **Job shop**.
- **Open shop**.

Oltre a tali strutture prototipali, si osservano varianti sul tema, e sono possibili **strutture ibride** (ad esempio **flexible flow shop**).

5.3 Misure di prestazione

Funzioni di penalità Si indica con C_i il **tempo di completamento** del job $i \in [n]$, ovvero il tempo di completamento della sua **ultima operazione**. Un tipico modo per costruire misure di prestazione è associare ad ogni job J_i un **termine di penalità** $\gamma_i(C_i)$. I termini tipicamente utilizzati sono

- **Tempo di completamento**: $\gamma_i(C_i) = C_i$.
- **Flow time**: $\gamma_i(C_i) = F_i = C_i - r_i$.
- **Lateness**: $\gamma_i(C_i) = L_i = C_i - d_i$.
- **Tardiness**: $\gamma_i(C_i) = T_i = \max\{C_i - d_i, 0\}$.
- **Earliness**: $\gamma_i(C_i) = E_i = \max\{d_i - C_i, 0\}$.
- **Indicatore di ritardo**: $\gamma_i(C_i) = U_i = \begin{cases} 1 & \text{se } C_i > d_i, \\ 0 & \text{se } C_i \leq d_i. \end{cases}$

Pesi e priorità dei job È possibile associare **pesi** w_i ai job, allo scopo di esprimere una **priorità**. Si può per esempio valutare per ogni job J_i la **tardiness pesata** $w_i T_i$.

Sulla base di questi "**mattoncini**" si possono costruire misure di prestazione **aggregate**, ottenendo funzioni del tipo: $\min \sum_{i=1}^{n} \gamma_i(C_i)$, $\min \max_{i=1,\dots,n} \gamma_i(C_i)$.

Principali misure di prestazione aggregate Le principali misure di prestazione aggregate sono

- **Flow time totale**: $\sum_i F_i$.
- **Flow time totale pesato**: $\sum_i w_i F_i$.
- **Massima lateness**: $L_{\max} = \max_i L_i$.
- **Tardiness totale pesata**: $\sum_i w_i T_i$.
- **Makespan**: $C_{\max} = \max_i C_i$.
- **Numero di job in ritardo**: $\sum_i U_i$.

Equivalenza tra misure di prestazione Alcune misure sono tra di loro **equivalenti**, nel senso che una soluzione ottima rispetto a una di esse è ottima anche per l'altra. Ad esempio, *la **lateness totale** è equivalente al **flow time totale***

$$\sum_{i=1}^{n} L_i = \sum_{i=1}^{n} C_i - \sum_{i=1}^{n} d_i = \sum_{i=1}^{n} F_i + \sum_{i=1}^{n} (r_i - d_i).$$

Le due misure differiscono per una **costante**. Inoltre, una soluzione ottima rispetto a $L_{\max}$ è ottima anche rispetto a $T_{\max}$ (ma non è assicurato il viceversa): $T_{\max} = \max\{L_{\max}, 0\}$.

Misure di prestazione regolari Le misure di prestazione finora elencate sono **funzioni non decrescenti** dei tempi di completamento; si parla in questo caso di **misure di prestazione regolari**. Nel caso di misure regolari, le informazioni di **tempistica** possono essere ricavate dal **sequenziamento**, in quanto ogni operazione è eseguita **al più presto** (**soluzioni attive**).

Misure di prestazione non regolari Nel caso di **misure non regolari**, ciò non è più vero e il problema risulta **più complesso**. Tali misure sono utili per tenere conto dell'**inopportunità di completare un job prima della due date**, ad esempio adottando come misura

$$\sum_{i=1}^{n}(w_i^T T_i + w_i^T E_i).$$

In questo caso esiste una regione in cui la funzione **decresce** al crescere del tempo di completamento, e **sequencing e timing non coincidono** e la soluzione ottima non è necessariamente **attiva o semiattiva**. Una schedulazione si dice **semiattiva** se, fissato il sequencing, ogni operazione è eseguita il più presto possibile senza violare l'ordine dei job su alcuna macchina. Mentre, una schedulazione è **attiva** se non esiste alcuna operazione che possa essere anticipata senza ritardarne almeno un'altra; ogni soluzione attiva è anche semiattiva, ma non viceversa.

Notazione di Graham Per classificare i problemi di schedulazione si utilizza la codifica a tre campi $\alpha|\beta|\gamma$ dovuta a **Graham**.

Campo α: layout delle macchine Il campo α descrive il **layout delle macchine**

- 1: macchina singola.
- P, Q, R: macchine parallele identiche, correlate e scorrelate.
- F: flow shop (ad esempio $F2$).
- J: job shop.

Campo β: vincoli aggiuntivi Il campo β descrive la presenza di eventuali **vincoli aggiuntivi**

- **res**: risorse aggiuntive.
- **prec**: vincoli di precedenza tra job.
- r_i: tempi di rilascio non nulli.
- $\bar{d}_i$: deadline hard.
- s_{ij}: setup dipendenti dalla sequenza.
- **perm**: flow shop a permutazione, la frequenza di job si mantiene su tutte le macchine.
- **no-wait**: assenza di buffer tra operazioni: ragioni tecnologiche impongono che le operazioni vengano eseguite consecutivamente senza interruzioni.

Campo γ: misura di prestazione Il campo γ descrive la **misura di prestazione** da ottimizzare.

5.4 Algoritmi di soluzione

Algoritmi polinomiali Esistono alcuni casi semplici risolvibili mediante **algoritmi polinomiali**

- $1//L_{\max}$: regola **EDD** che ordina i job in ordine di **due date** crescente.
- $1//\sum_i w_i c_i$: regola **WSPT** (weigthed shortest processing time) che ordina i job in ordine decrescente del rapporto w_i/p_i.
- $F2//C_{\max}$: algoritmo di **Johnson** (in questo caso la soluzione ottima applica la stessa sequenza sulle due macchine).

Problemi NP-hard ed euristiche In generale, i problemi di schedulazione sono **NP-hard** e vengono affrontati mediante **euristiche**, sia costruttive sia iterative (**metaeuristiche** basate su ricerca locale). Le **regole di priorità** costituiscono una strategia costruttiva di base.

Regola ATC (Apparent Tardiness Cost) Esistono anche regole non banali, come la **regola ATC (Apparent Tardiness Cost)**, proposta per problemi $J//\sum_i w_i T_i$. L'espressione della priorità del job J_i sulla macchina M_j al tempo t è

$$\frac{w_i}{p_{ij}} \exp\left(- \left[\frac{d_i - t - p_{ij} - \sum_{q=j+1}^{m_i} (W_{iq} + p_{iq})}{k\bar{p}} \right]^+ \right),$$

dove w_i è il peso del job J_i e p_{ij} è il suo tempo di processo sulla macchina M_j; W_{iq} è una stima del tempo di attesa di J_i sulla macchina M_q; m_i è il numero di macchine che il job deve ancora visitare; k è un parametro da selezionare e $\bar{p}$ è il tempo di processo medio dei job in coda.

Interpretazione del termine di slack L'espressione

$$t^* = d_i - p_{ij} - \sum_{q=j+1}^{m_i} (W_{iq} + p_{iq}),$$

può essere interpretata come il *massimo tempo di inizio* di J_i su M_j, ottenuto sottraendo un lead time stimato dalla due date d_i. Se $t \leq t^*$ si può ritenere di essere **in tempo**, altrimenti si corre il rischio di arrivare in ritardo. **Se si è in tempo**, il termine tra parentesi quadre è *positivo*, e la priorità cresce esponenzialmente al crescere di t.

Quando si è in ritardo, il termine tra parentesi quadre è *negativo*, per cui l'argomento dell'esponenziale è zero. La priorità è quindi w_i/p_{ij}, che coincide con la regola WSPT.

Relazione con la tardiness totale pesata *Se molti job sono in ritardo, la tardiness totale pesata diventa una funzione obiettivo quasi equivalente al tempo di completamento totale pesato*

$$\sum_i w_i T_i = \sum_i w_i \max\{C_i - d_i, 0\} \approx \sum_i w_i C_i - \sum_i w_i d_i.$$

Il problema di minimizzazione di $\sum_i w_i C_i$ è risolto, su macchina singola, dalla **regola WSPT**.

Lookahead e ricerca locale La **miopia** delle regole di priorità può essere ridotta introducendo un certo grado di **lookahead**, ad esempio mediante strategie di *beam search*. Una vasta classe di algoritmi iterativi si basa su perturbazioni locali della soluzione corrente, che definiscono una struttura di vicinato. I principali problemi sono

- Sfuggire a **minimi locali** (tabu search, algoritmi genetici, questi richiedono attenta codifica e decodifica delle soluzioni).
- Esplorare **grandi vicinati** in maniera efficiente (LNS, Large Neighborhood Search, utile anche per misure non regolari).
- Evitare la **creazione di cicli**.

Molte di tali questioni richiedono una formulazione mediante un **grafo disgiuntivo**.

Grafi disgiuntivi In un **grafo disgiuntivo**, i nodi rappresentano le operazioni, con l'aggiunta di nodi dummy iniziali e finali. Gli **archi congiuntivi** rappresentano vincoli di precedenza tecnologici, associati ai cicli di ciascun job. Gli **archi disgiuntivi** vanno orientati e rappresentano i vincoli di capacità per ogni macchina (una clique per macchina). Il **cammino critico**, ossia il cammino di lunghezza massima dal nodo iniziale al nodo finale, fornisce il makespan.

5.5 Modello MILP per $J//C_{max}$

Formulazione Sulla base del grafo disgiuntivo possiamo costruire un modello MILP per il problema $J//C_{max}$. Sia $N = \{0, 1, \ldots, N-1, N\}$ l'insieme dei nodi, dove 0 e N sono nodi dummy. Sia P l'insieme degli archi congiuntivi e D l'insieme degli archi disgiuntivi. La variabile binaria x_{ij}, se posta pari a 1, indica che l'operazione i precede l'operazione j.

$$
\begin{aligned}
\min \quad & C_N \\
\text{s.t.} \quad & C_j \geq C_i + p_j, && \forall (i,j) \in P, \\
& C_j \geq C_i + p_j - M(1 - x_{ij}), && \forall (i,j) \in D, \\
& C_i \geq C_j + p_i - M x_{ij}, && \forall (i,j) \in D, \\
& x_{ij} \in \{0,1\}, && \forall (i,j) \in D, \\
& C_i \geq p_i, && \forall i \in N.
\end{aligned}
$$

Il modello può essere adattato ad altre misure di prestazione, ma non è trattabile in pratica a causa di bound deboli. Tuttavia costituisce una base utile per metaeuristiche e approcci LNS (*destroy and repair*).

Il ruolo del cammino critico In una strategia di ricerca locale, perturbazioni arbitrarie delle sequenze possono creare cicli. Nel problema $J//C_{\max}$, *se si perturbano archi disgiuntivi appartenenti al cammino critico, non si possono creare cicli, e tali perturbazioni sono le sole utili.*

5.6 Procedura Shifting Bottleneck

Idea generale La procedura **shifting bottleneck**, originariamente pensata per il problema $J//C_{\max}$, sfrutta il grafo disgiuntivo per *decomporre il problema* in una sequenza di problemi del tipo $1/r_i/L_{\max}$, per i quali sono disponibili algoritmi *branch-and-bound* efficienti. Se una macchina M_b è il **collo di bottiglia**, le altre macchine possono essere trattate come *risorse a capacità infinita*, eliminando i relativi archi disgiuntivi corrispondenti alle altre macchine e mantenere solo quelle corrispondenti alle altre macchine.

Teste e code nel grafo disgiuntivo Si consideri il nodo O_{ib} del grafo, corrispondente all'operazione del job J_i sulla macchina M_b, e sia C_{ib} il suo tempo di completamento. Si consideri il **cammino critico** dal nodo fittizio iniziale al nodo O_{ib}. La sua lunghezza h_{ib} è la somma dei tempi di processo delle operazioni di J_i che precedono l'operazione O_{ib}, e ne costituisce la **testa**, ovvero il **tempo di rilascio** dell'operazione O_{ib}. In modo analogo si definisce la **coda** t_{ib} come la lunghezza del cammino critico dal nodo O_{ib} al nodo fittizio finale. In questo caso t_{ib} è la somma dei tempi di processo delle operazioni che seguono O_{ib}.

Approssimazione del makespan Si definisce una data di consegna per il job J_i sulla macchina M_b, approssimando il makespan come $C_{\max} = \max_i\{C_i\} \approx \max_i\{C_{ib} + t_{ib}\}$. Se si sottrae una costante arbitraria K dalla misura di prestazione, la soluzione ottima del problema non cambia, ottenendo il problema equivalente

$$\max_i\{C_{ib} + t_{ib}\} - K = \max_i\{C_{ib} - (K - t_{ib})\} = \max_i\{C_{ib} - d_{ib}\},$$

dove è stata introdotta una **due date locale** $d_{ib} = K - t_{ib}$. Più lunga è la coda t_{ib}, più stretta è la data di consegna d_{ib}.

Riduzione a $1/r_i/L_{max}$ Il problema $J//C_{\max}$ si riconduce quindi a un problema $1/r_i/L_{\max}$, in cui: $r_{ib} = h_{ib}$, $d_{ib} = K - t_{ib}$. I tempi di rilascio sono dati dalle **teste**, mentre le date di consegna dipendono dalle **code**.

Identificazione del collo di bottiglia Un modo ragionevole per identificare il **collo di bottiglia** consiste nel risolvere i problemi $1/r_i/L_{\max}$ per tutte le macchine, definendo per ciascuna le teste e le code delle operazioni. Il collo di bottiglia è la macchina che presenta il valore di $L_{\max}$ peggiore. Dopo avere risolto i problemi su macchina singola, si individua il collo di bottiglia M_b e la relativa sequenza. È quindi possibile orientare gli archi disgiuntivi corrispondenti a M_b e rischedulare le altre macchine. In pratica, si risolvono i problemi su macchina singola per le rimanenti $M-1$ macchine, aggiornando teste e code. Il procedimento prosegue fino ad avere schedulato tutte le macchine, congelandole una per volta. Sono possibili *raffinamenti iterativi*, e la strategia si estende facilmente al problema $J/r_i/L_{max}$; sono state proposte anche ulteriori estensioni.

6 CAP 6 – Fondamenti microeconomici del pricing e modelli di scelta discreta

Driver del profitto Il profitto è determinato da quattro fattori fondamentali ed è descritto dal seguente modello semplice: Profitto = (Prezzo − Costo variabile) · Volume − Costo fisso.

Sensibilità del profitto ai fattori *Quale sarebbe l'impatto di una variazione di ciascun fattore, supponendo che gli altri rimangano invariati?* Si considerano variazioni percentuali dei fattori nell'intervallo da −10% a +10% e il loro impatto percentuale sul profitto. Un aumento del prezzo del 10% comporta un raddoppio del profitto. Al contrario, una riduzione del costo variabile del 10% implica un aumento del profitto pari al 50%. Gli altri due fattori hanno un impatto minore. Il grafico risulta simmetrico rispetto alle variazioni positive e negative.

Osservazioni *È importante notare che la modifica del prezzo è essenzialmente istantanea e priva di costo, a differenza degli investimenti in pubblicità o riduzione dei costi.* Tuttavia, il **modello è poco realistico** poiché il prezzo influisce sul volume e il costo non è necessariamente una funzione affine del volume. È quindi necessario considerare le interazioni tra i fattori, pur utilizzando questo modello grezzo per sviluppare intuizioni.

Catene causali È necessario **modellare il legame tra prezzo e volume** (domanda) ed essere consapevoli delle possibili catene causali

- Prezzo → Volume → Ricavi → Profitto.
- Prezzo → Volume → Costi → Profitto.
- Prezzo → Prezzi dei concorrenti → Quota di mercato → Volume → Ricavi → Profitto.
- Prezzo del produttore → Prezzo del rivenditore → Volume → Ricavi → Profitto.

Pricing e revenue management L'**obiettivo** di un'impresa viene comunemente identificato nella **massimizzazione del profitto**. Tuttavia, l'analisi precedente mostra che ciò non si riduce alla sola minimizzazione dei costi.

Se il profitto è ricavo meno costo, si aprono ulteriori possibilità. Le strategie di *revenue e yield management* sono diventate comuni a partire dall'industria aerea.

Strategie di pricing Le **principali strategie di pricing** includono

- La strategia di pricing apparentemente più ovvia è quella **basata sui costi**: si valuta il **costo** di un prodotto e si applica un **mark-up**. Questo approccio semplice **non tiene conto** della reazione dei **consumatori** e dei **competitor**.

- Un'altra idea, piuttosto comune in **economia**, è che il **prezzo** derivi dall'**equilibrio tra domanda e offerta**. Questo porta al concetto di **pricing basato sulla domanda**, che può essere affrontato tramite **modelli semplici** nel caso di **monopolio**.
- Un ulteriore elemento è il **ruolo della concorrenza**. Nella letteratura classica di **microeconomia**, questo aspetto può essere analizzato tramite **modelli di teoria dei giochi**. Tali modelli possono essere utilizzati anche per descrivere l'**interazione tra impresa e consumatore**, e non solo tra imprese.

Tipologie di interazione Le **forme di competizione e interazione** sono molteplici

- Competizione tra imprese (oligopolio vs. monopolio).
- Interazioni lungo la supply chain (produttori e rivenditori).
- Comportamento strategico dei clienti.

Pricing nella pratica È necessario un cambio di paradigma: Design–Build–Price → Price–Design–Build.

Occorre selezionare una strategia di posizionamento coerente: luxury, premium, medium, low, ultra-low. La strategia di pricing dipende dagli attributi chiave del prodotto o servizio: funzionali, emotivi, simbolici, etici.

Scelta del modello È necessario scegliere il modello appropriato in funzione del contesto

- Acquisti ripetuti vs. acquisti singoli.
- Beni di prima necessità vs. beni voluttuari.
- Business-to-business (B2B) vs. business-to-consumer (B2C).

Meccanismi alternativi Il prezzo non è l'unico strumento per migliorare profitto e ricavi. Esistono diversi meccanismi alternativi:

- **Bundling e tying**, sconti di quantità, pacchetti e servizi ancillari; talvolta unbundling.
- Utilizzo di **coupon**.
- **Abbonamenti** personalizzati.
- Gestione della disponibilità di **classi differenti**.
- **Overbooking** per gestire no-show e capacità non stoccabile.
- **Condivisione del rischio** e progettazione di contratti e incentivi.

6.1 Pricing e funzioni di domanda

Un modello introduttivo: relazione lineare tra domanda e prezzo La domanda è una variabile casuale, influenzata da una molteplicità di fattori, incluso (ma non solo) il prezzo. Come punto di partenza si introduce un **modello semplice** basato su una **funzione di domanda lineare** che lega prezzo e domanda

$$d(p) = \alpha - \beta p.$$

Interpretazione dei parametri Un'ipotesi apparentemente ovvia è $\beta > 0$, ma esistono controesempi forniti dai beni di lusso e dai casi in cui il **prezzo** *funge da segnale di qualità*. L'intercetta α rappresenta la **domanda** quando $p = 0$, un livello di prezzo per il quale il modello ha poco senso. In altri modelli, la domanda tende all'infinito per $p \to 0$, ma se la popolazione dei consumatori è finita, α può essere interpretato come la dimensione della popolazione.

Dominio di validità Poiché la domanda non può essere negativa, il modello ha senso nell'intervallo di prezzo $p \in [0, p_{\text{lim}}]$, dove si definisce il prezzo limite $p_{\text{lim}} = \frac{\alpha}{\beta}$ come il prezzo massimo per il quale la domanda è positiva.

Forma troncata della funzione di domanda Il modello può essere riscritto in una forma potenzialmente migliore come

$$d(p) = (\alpha - \beta p)^+.$$

Funzione di domanda inversa In microeconomia è comune **invertire la funzione di domanda** per ottenere il prezzo di mercato quando una quantità q viene prodotta e offerta sul mercato. Invertendo l'equazione $d(p) = \alpha - \beta p$, si ottiene la **funzione di domanda inversa**

$$p(q) = a - bq,$$

dove $a = \alpha/\beta$ e $b = 1/\beta$. Questa forma è utile nello studio della *competizione basata sulle quantità nella microeconomia elementare*, ma risulta meno naturale nel contesto di una modellazione statistica appropriata per il pricing management. Tuttavia, esistono mercati in cui il prezzo emerge da meccanismi complessi (ad esempio aste) che dipendono dalla disponibilità totale.

Massimizzazione dei ricavi e del profitto Un **obiettivo** naturale è la **massimizzazione del profitto**. Nella sua forma più semplice, il *profitto* è dato da *ricavi meno costi*, e può essere espresso come funzione del prezzo o della quantità. La seconda scelta è più naturale per esprimere i costi

$$\pi(q) = p(q) \cdot q - c(q) = r(q) - c(q),$$

dove la funzione di costo $c(q)$ è una funzione generica della quantità e può tenere conto di costi fissi, economie o diseconomie di scala. Si introduce inoltre la **funzione di ricavo** $r(q) := p(q) \cdot q$.

Condizione di ottimalità Assumendo che tutte le funzioni siano differenziabili e che la funzione di profitto complessiva sia *concava*, si applica la **condizione di ottimalità del primo ordine**

$$\pi'(q^*) = r'(q^*) - c'(q^*) = 0 \quad \Rightarrow \quad r'(q^*) = c'(q^*).$$

La **quantità ottimale** è tale per cui il ricavo marginale e il costo marginale coincidono. Se il ricavo marginale è maggiore del costo marginale, il beneficio in termini di ricavi derivante da un aumento della produzione supera l'aumento dei costi, e conviene aumentare q. Se il costo marginale è maggiore del ricavo marginale, conviene ridurre q.

Nel caso di **funzione di domanda inversa** lineare e costo variabile costante, il *profitto è una funzione quadratica concava*

$$\pi(q) = (a - bq) \cdot q - cq = (a - c) \cdot q - bq^2,$$

e la *condizione di ottimalità del primo ordine* è: $\pi'(q) = (a - c) - 2bq$, da cui si ottiene

$$q^* = \frac{a - c}{2b}.$$

Osservazioni La massimizzazione del profitto non è equivalente alla minimizzazione dei costi. La minimizzazione dei costi è appropriata quando una certa domanda deve comunque essere soddisfatta e occorre scegliere tra diverse tecnologie produttive o piani di produzione nel tempo. In alcuni settori il costo marginale di un'unità aggiuntiva è trascurabile (industria aerea, servizi web) o il costo di produzione è un costo sommerso (markdown management). In questi casi può essere sensata la massimizzazione dei ricavi, naturalmente espressa come funzione del prezzo

$$\max r(p) = p \cdot d(p) = p \cdot (\alpha - \beta p) = \alpha p - \beta p^2.$$

La **soluzione ottima per la massimizzazione dei ricavi** è

$$p^* = \frac{\alpha}{2\beta} \quad \Rightarrow \quad d^* = \frac{\alpha}{2},\ r^* = \frac{\alpha^2}{4\beta}.$$

Interpretazione geometrica La soluzione può essere **interpretata geometricamente**: il prezzo ottimale è il *punto medio* dell'intervallo dei prezzi e massimizza l'area del rettangolo associato ai ricavi.

Stima del modello Nella microeconomia elementare si utilizzano **modelli deterministici**, ma anche un modello di domanda lineare apparentemente banale come $d(p) = \alpha - \beta p$ contiene parametri ignoti che devono essere stimati.

Apprendimento e sperimentazione Anche assumendo che il modello lineare sia corretto, esiste un problema rilevante: il costo della procedura di stima. *È necessario apprendere online piuttosto che offline, e la pianificazione degli esperimenti è cruciale. Quali prezzi utilizzare per apprendere efficacemente i parametri?* L'attenzione dovrebbe concentrarsi sulla pendenza β, poiché l'intercetta α è principalmente un costrutto matematico.

L'**errore standard della stima** della pendenza è dato da

$$SE_\beta = \frac{\sigma_\varepsilon}{\sqrt{\sum_{k=1}^{n}(p_k - \bar{p})^2}},$$

dove σ_ε è la deviazione standard (ignota) degli errori. *Questo mostra che maggiore è la dispersione dei prezzi sperimentati, migliore è l'apprendimento.* Tuttavia, prezzi molto bassi o molto elevati possono danneggiare i ricavi, generando un trade-off tra apprendimento e costo dell'apprendimento.

Problemi pratici aggiuntivi Nella pratica emergono ulteriori difficoltà. In primo luogo, non tutti i **prezzi commerciali** sono effettivamente applicabili. Inoltre, **grandi variazioni di prezzo** tendono a enfatizzare la **non linearità** del comportamento della domanda. Un altro aspetto rilevante è il **ruolo del tempo**: nei **beni di moda** il tempo è un fattore essenziale e il **periodo di apprendimento** risulta spesso limitato. Si aggiungono poi i **fattori confondenti**, poiché una variazione della domanda può dipendere da elementi diversi dal prezzo stesso. Vi sono anche **problemi operativi ed esecutivi**, come ad esempio la necessità di modificare i cartellini dei prezzi in un negozio fisico. Infine, una **sperimentazione eccessiva** sui prezzi può generare **insoddisfazione nei consumatori** e indurre **comportamenti strategici**.

La funzione di costo È comune considerare una componente di costo fisso e una variabile; la forma più semplice di costo variabile è **lineare**

$$c(q) = F + cq.$$

Una **funzione di costo alternativa** è $c(q) = \begin{cases} F + cq & \text{se } q > 0, \\ 0 & \text{se } q = 0, \end{cases}$ che può essere scritta come

$$c(q) = F \cdot \delta(q) + cq,$$

dove $\delta(q) = 1$ se $q > 0$ e $\delta(q) = 0$ altrimenti. Per evitare ambiguità, si parla di **costo fisso** nel primo caso e di **carica fissa** nel secondo. Si noti che questa distinzione può dipendere dalla scala temporale e dal livello decisionale gerarchico: alla scala temporale appropriata, tutti i costi sono variabili.

Costo marginale e costo medio Data una funzione di costo, si definiscono il **costo marginale** $c'(q)$ e il **costo medio** $\dfrac{c(q)}{q}$.

Regolarità della funzione di costo Per definire il costo marginale è necessario assumere la *differenziabilità*. Oltre alle cariche fisse, discontinuità possono essere introdotte da sconti di quantità all-units, mentre sconti incrementali introducono punti angolosi. Quando la produzione è discreta e q assume valori interi, il costo marginale può essere approssimato come $c(q+1) - c(q)$.

Economie e diseconomie di scala Il **costo marginale** può essere crescente o decrescente. Quando il costo marginale è decrescente, assumendo la differenziabilità, si ha $c''(q) < 0$, e quindi la funzione di costo è concava. Questo modella un'**economia di scala**, ossia un aumento dell'efficienza all'aumentare della produzione. Al contrario, una funzione di costo convessa, caratterizzata da un costo marginale crescente, modella una **diseconomia di scala**.

Funzioni di domanda Esiste una grande varietà di modelli di domanda

- Si può **modellare** la domanda aggregata dell'intero mercato, di un canale selezionato o di una parte del mercato, oppure la domanda di un singolo individuo.
- Si possono considerare acquisti ripetuti oppure no, a seconda della natura del bene o servizio (beni stock vs. beni flusso) e dell'orizzonte temporale.
- Si può considerare o meno l'**incertezza**.

I **fattori di input** possono includere tempo, vendite passate, prezzi dei concorrenti o di beni alternativi. L'**output** del modello può essere una variabile

- **Reale** (domanda aggregata, eventualmente approssimante un valore intero elevato).
- **Intera** (quantità acquistata da un singolo consumatore).
- **Binaria** (acquisto o non acquisto).
- **Categorica** o un vettore (bundle o portafoglio di beni acquistati).

La funzione di domanda lineare Per scopi principalmente illustrativi, si è già considerata la funzione di domanda lineare: $d(p) = (\alpha - \beta p)^+$.

È naturale interrogarsi sulla **validità di un modello così semplice**

- Come può essere giustificato dal punto di vista economico? È necessario indagare una fondazione microeconomica per il modello di domanda.
- È un modello validato empiricamente o è contraddetto dal buon senso e dal comportamento reale dei consumatori?

Assumendo la *differenziabilità*, una caratteristica fondamentale di ogni funzione di domanda è la sua **pendenza** $d'(p)$, che per buon senso dovrebbe essere *negativa*, come nel modello lineare. In effetti è solitamente negativa, ma esistono eccezioni: segnale di qualità e beni di lusso e consumo vistoso.

Possono quindi esistere intervalli di prezzo per i quali la pendenza è *positiva*, un fatto confermato da evidenze empiriche.

Funzioni di domanda: elasticità puntuale Poiché il valore della pendenza dipende dalla scala, ovvero dalle unità di misura della domanda, una misura più appropriata è l'**elasticità di prezzo della domanda**

$$\frac{\delta d/d}{\delta p/p} = \frac{\delta d}{\delta p} \cdot \frac{p}{d},$$

facendo tendere $\delta p \to 0$, si ottiene l'**elasticità puntuale**

$$\varepsilon(p) = -\frac{d'(p)p}{d(p)},$$

dove il segno *meno* è introdotto per comodità, poiché la pendenza è tipicamente negativa.

Le *funzioni di domanda* possono essere classificate come

- **Elastiche**, quando $\varepsilon(p) > 1$ (perfettamente elastiche se pari a ∞).
- A **elasticità unitaria**, quando $\varepsilon(p) = 1$.

- **Anelastiche**, quando $\varepsilon(p) < 1$ (perfettamente anelastiche se pari a 0).

La natura della domanda può dipendere dal punto specifico considerato.

Esempio: funzione di domanda lineare *L'elasticità non deve essere confusa con la pendenza.* Il modello di domanda lineare presenta una pendenza costante, ma ciò non implica un'elasticità puntuale costante.

Per una funzione di domanda lineare, l'**elasticità** è $\varepsilon(p) = \frac{\beta p}{\alpha - \beta p}$. Essa cresce da zero a $+\infty$ nell'intervallo $[0, p_{\text{lim}}]$. L'elasticità è pari a 1 per

$$p = \frac{\alpha}{2\beta} = p^*,$$

ossia nel *punto medio dell'intervallo dei prezzi*. La domanda è **anelastica** per $p < p^*$ ed **elastica** per $p > p^*$. Nel limite, quando $p \to p_{\text{lim}} = \alpha/\beta$, diventa **perfettamente elastica**: una piccola riduzione di prezzo fa passare la domanda da $d = 0$ a $d > 0$. *L'elasticità non è costante per una funzione di domanda lineare.*

Esempio: funzione di domanda a elasticità costante È naturale chiedersi *se esista una funzione di domanda con elasticità costante*. Imponendo $\frac{d'(p)p}{d(p)} = -\varepsilon$, si ottiene l'equazione differenziale ordinaria $d'(p) = -\varepsilon \cdot \frac{d(p)}{p}$, che ammette come soluzione $d(p) = c \cdot p^{-\varepsilon}$, dove $c = d(1)$.

Stima tramite trasformazione logaritmica È possibile derivare un *legame tra domanda a elasticità costante e domanda lineare* mediante una trasformazione dei dati basata sui logaritmi

$$d = cp^{-\varepsilon} \;\Rightarrow\; \log d = \log c - \varepsilon \log p.$$

In questo modo, gli strumenti standard di regressione lineare possono essere utilizzati per stimare il modello. *L'uso dei logaritmi è utile per semplificare modelli basati su effetti moltiplicativi anziché additivi.*

In termini di incrementi, per $x > 0$ vale

$$\frac{d \log x}{dx} = \frac{1}{x} \quad \Rightarrow \quad \delta \log x \approx \frac{\delta x}{x}.$$

Di conseguenza, l'**elasticità** può essere riscritta come

$$\frac{\delta d/d}{\delta p/p} \approx \frac{d \log d(p)}{d \log p}.$$

Nel caso di **domanda a elasticità costante**, il **ricavo** è $R(p) = p \cdot cp^{-\varepsilon} = cp^{1-\varepsilon}$. Il ricavo è costante nel caso di elasticità unitaria, crescente nel caso di domanda anelastica e decrescente nel caso di domanda elastica. *Pertanto, una riduzione di prezzo aumenta i ricavi solo se la domanda è elastica.*

Interpretare i modelli di domanda: willingness-to-pay Come si possono confrontare diversi modelli di domanda? La funzione di domanda lineare è un modello sensato? Un **modello lineare** può essere (nel migliore dei casi) un'**approssimazione locale**, ma è necessario comprendere più a fondo la razionalità economica alla base di tale modello. Ciò significa che occorre capire che cosa determina realmente la domanda.

Un meccanismo di base dietro i modelli di domanda è il concetto di **prezzo di riserva o willingness-to-pay**: ogni consumatore è caratterizzato dal prezzo massimo al quale è disposto ad acquistare un bene. In generale, si può produrre una *funzione di domanda a tratti costante*.

L'intuizione suggerisce che, nel limite continuo, assumendo una willingness-to-pay uniformemente distribuita, si ottenga una funzione di domanda lineare. Funzioni di domanda alternative possono essere definite assumendo *distribuzioni differenti* del prezzo di riserva e studiando la sensibilità della domanda rispetto al prezzo.

Surplus del consumatore e discriminazione di prezzo Si consideri un consumatore disposto a pagare 100 per un bene. Se il prezzo è 70, si dice che il **consumer surplus** è la differenza tra willingness-to-pay e prezzo pagato. Quando viene applicato un prezzo unico, una parte di consumer surplus è ottenuta dall'insieme dei consumatori che acquistano.

Geometricamente, ciò è visualizzabile come il triangolo CS, con area

$$CS = \frac{1}{2}\left(\frac{\alpha}{\beta} - p_0\right)(\alpha - \beta p_0) = \frac{1}{2\beta}(\alpha - \beta p_0)^2,$$

dove p_0 è il **prezzo scelto**. Si noti che il consumer surplus è nullo quando $p_0 = p_{\text{lim}} = \alpha/\beta$, ed è pari a $\alpha^2/(2\beta)$, ossia l'area totale del triangolo, quando $p_0 = 0$. Il **ricavo raccolto** è rappresentato dall'area rettangolare REV.

In *microeconomia*, il consumer surplus è una misura approssimata dell'utilità del consumatore derivante dallo scambio. Guardando dalla prospettiva opposta, il consumer surplus è, in un certo senso, ricavo perso per l'impresa. Lo stesso vale per il triangolo LR, che rappresenta **ricavo perso** dovuto ai consumatori che non acquistano perché p_0 eccede la loro willingness-to-pay.

Considerando solo i ricavi, *l'obbligo di scegliere un solo prezzo genera perdite su entrambi i lati*: consumer surplus (opportunità mancata di profitto maggiore) e vendite perse. Idealmente, l'impresa vorrebbe applicare un prezzo diverso a ciascun individuo, pari alla sua willingness-to-pay (assumendo costo marginale nullo).

Naturalmente, fissare un prezzo individuale non è pratico, per non dire apertamente illegale. Come può un'impresa estrarre almeno parzialmente consumer surplus? La chiave è la discriminazione di prezzo basata sulla segmentazione dei consumatori.

Esempio di discriminazione di prezzo Applicando discriminazione di prezzo, l'impresa riesce a estrarre consumer surplus. D'altra parte, una sottopopolazione che sarebbe esclusa dallo scambio può partecipare. In generale, *non è detto che si ottenga una soluzione win–win*. Inoltre, *potrebbe non essere possibile discriminare in modo netto*. Esistono **diversi tipi di discriminazione**

- **Discriminazione completa**: ogni consumatore paga un prezzo specifico.
- **Segmentazione diretta**: segmenti identificabili pagano prezzi diversi.
- **Segmentazione indiretta**: si offrono varianti di prodotto/servizio e i consumatori scelgono; ciò funziona anche con caratteristiche non identificabili.

Formalizzare willingness-to-pay e consumer surplus Dopo un'introduzione informale, si passa a un quadro più formale. La **willingness-to-pay è il prezzo massimo (prezzo di riserva) che un consumatore è disposto a pagare per un bene**. La distribuzione della willingness-to-pay nella popolazione determina la forma della *funzione di domanda*.

Si consideri un modello continuo e si definisca $w(x)$ come densità della willingness-to-pay. Essa svolge un ruolo simile alla densità di una variabile casuale, nel senso che

$$\int_{p_1}^{p_2} w(x)\,dx$$

è la frazione di popolazione con willingness-to-pay tra p_1 e p_2. Quindi

$$d(p) = D_{\max} \int_{p}^{+\infty} w(x)\,dx,$$

dove $D_{\max} = d(0)$ è la **domanda massima ottenibile** (assumendo una popolazione finita). Poiché $d'(p) = -D_{\max}\, w(p)$, si può legare domanda e willingness-to-pay tramite

$$w(p) = -\frac{d'(p)}{D_{\max}}.$$

Surplus del consumatore aggregato Se un consumatore disposto a pagare p_1 paga invece $p_0 < p_1$, gode di un consumer surplus $p_1 - p_0$. Dato $w(x)$, si integra il surplus al prezzo p_0 sulla popolazione che acquista e si definisce il **consumer surplus totale (netto)**

$$S(p_0) = D_{\max} \int_{p_0}^{+\infty} w(x)\,(x - p_0)\,dx - \int_{p_0}^{+\infty} d'(x)\,(x - p_0)\,dx = -\int_{p_0}^{+\infty} d'(x)x\,dx + p_0 \int_{p_0}^{+\infty} d'(x)\,dx$$

$$= -\left[d(x)\,x|_{p_0}^{+\infty} - \int_{p_0}^{+\infty} d(x)\,dx\right] - p_0 d(p_0) = \int_{p_0}^{+\infty} d(x)\,dx.$$

Quindi, il **consumer surplus** è *l'area sotto la curva di domanda*, a destra del prezzo applicato p_0.

Obiettivo di estrarre consumer surplus Un **obiettivo naturale** per l'impresa è **"estrarre" consumer surplus** almeno parzialmente tramite una forma di discriminazione di prezzo. *Se l'impresa è in grado di discriminare, migliorerà il proprio profitto.* L'intuizione suggerisce che ciò avvenga necessariamente a spese dei consumatori. L'esempio seguente mostra che non è necessariamente così, se consumatori con willingness-to-pay minore possono acquistare a un prezzo minore.

La funzione logit **Integrare la densità di willingness-to-pay**, analogamente a quanto si fa con una PDF per ottenere una CDF, **fornisce una funzione di domanda**. Un modello di domanda lineare non deriva da una distribuzione sensata della willingness-to-pay; è ragionevole ipotizzare una funzione "a campana" con un massimo (*moda*) concentrato attorno al prezzo di riserva più tipico.

Nella letteratura di statistica e marketing si utilizzano due funzioni per questo e altri scopi, che portano a modelli probit e logit. I modelli **probit** derivano dalla CDF di una normale standard, mentre i modelli **logit** si basano sulla funzione logistica (sigmoide).

Definizione della funzione logistica La **funzione logistica** è definita come

$$L(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x},$$

e presenta la classica forma a S. Si osservi che $L(0) = 0.5$ e che la funzione logit ha la *proprietà di simmetria*: $L(-x) = 1 - L(x)$.

La **derivata** è

$$L'(x) = \frac{e^x}{(1 + e^x)^2} = L(x)\,(1 - L(x)).$$

La derivata ha la *proprietà di simmetria* $L'(x) = L'(-x)$ e il suo massimo si ha in $x = 0$.

Poiché le funzioni di domanda devono essere *decrescenti*, si deve ribaltare l'asse x da sinistra a destra e introdurre *fattori di scala* e traslazione sensati nella funzione logit, ottenendo la seguente funzione di domanda logit (decrescente)

$$d(p) = \frac{c\,e^{-(\alpha+\beta p)}}{1 + e^{-(\alpha+\beta p)}},$$

dove β, $c > 0$. Più grande è β, maggiore è la sensibilità al prezzo e la funzione è più ripida per $\alpha + \beta p^* = 0 \;\Rightarrow\; p^* = -\frac{\alpha}{\beta}$, il che suggerisce $\alpha < 0$. Questo è il valore per cui la densità di willingness-to-pay è massima e può essere considerato un "**prezzo di mercato**".

La **willingness-to-pay** corrispondente è

$$w(x) = -\frac{d'(p)}{d(0)} = \frac{K\,e^{-(\alpha+\beta x)}}{\left(1 + e^{-(\alpha+\beta x)}\right)^2},$$

dove $K = \beta c / d(0)$.

Esempio: un modello ibrido In alcuni casi estremi, il prezzo di un bene potrebbe anche essere negativo per incrementare le vendite del bene complementare. Questo può sembrare strano ma, in realtà, alcuni beni possono essere venduti sotto costo per ottenere ricavi dai beni complementari. Ad esempio, una stampante laser permette di vendere cartucce toner, e un rasoio permette di vendere lame.

Una pratica comune nel revenue management è l'**inventory rationing**, ossia la restrizione delle vendite. Questo è lo strumento principale nel revenue management basato sulle quantità, come si vedrà. Il *disaccoppiamento* tramite razionamento fornisce gradi di libertà aggiuntivi.

L'idea del **modello ibrido** è che prezzo e quantità non siano necessariamente variabili decisionali alternative. Nella **prima formulazione**, si assume implicitamente che la domanda coincida con le vendite: fissati i prezzi, le quantità vendute sono determinate in modo automatico dalla funzione di domanda. Questo porta però a soluzioni che, pur essendo ottimali dal punto di vista dei prezzi, possono risultare **non realistiche** in presenza di **vincoli di capacità**, perché la domanda generata potrebbe non essere fisicamente realizzabile.

Nella **seconda formulazione**, invece, si introduce un **disaccoppiamento tra domanda e vendite**. La funzione di domanda non determina più direttamente le quantità vendute, ma fornisce un **vincolo superiore** sulle quantità offerte. Questo consente di applicare il **razionamento delle scorte**, limitando le vendite anche quando la domanda sarebbe più elevata. In questo modo, il modello acquisisce **gradi di libertà aggiuntivi**, permettendo di combinare decisioni di prezzo e di quantità in modo più coerente con i vincoli operativi.

6.2 Modelli di teoria dei giochi

Problemi decisionali con più decisori Talvolta le decisioni di pricing non possono essere prese ignorando il contesto complessivo. In un contesto B2C, i consumatori strategici sono un problema. Le interazioni strategiche sono ancora più rilevanti in un contesto B2B, dove i produttori possono essere collegati da una supply chain in cui beni intermedi vengono scambiati e trasformati, oppure dove produttori e retailer che distribuiscono beni finali interagiscono. In tale contesto, i prezzi possono svolgere un ruolo importante come strumento per coordinare azioni e condividere rischi. Infine, che dire della competizione tra imprese? Occorre distinguere tra competizione in termini di prezzi o di quantità, e tra beni/servizi identici o differenziati. Questi problemi possono essere (almeno parzialmente) affrontati con **modelli di teoria dei giochi**.

Ipotesi semplificative Per semplicità, si considerano *giochi stilizzati*

- Ci sono solo due decisori (giocatori); ciascun giocatore ha un **obiettivo** (payoff) che vuole massimizzare e non esiste cooperazione.
- Ciascun giocatore prende **una sola decisione**; quindi non si considerano giochi sequenziali con decisioni multiple nel tempo.
- Si assume **informazione completa** e common knowledge: non c'è incertezza sui dati del problema né sui meccanismi che mappano decisioni in payoff; i due giocatori condividono la stessa visione del mondo e le regole del gioco, conoscono gli incentivi della controparte e ciascuno sa che l'altro possiede tutte le informazioni rilevanti.

Per avvicinarsi a una formalizzazione, si consideri il problema

$$\begin{aligned} &\max \pi_1(x_1, x_2) + \pi_2(x_1, x_2) \\ &\text{s.t. } x_1 \in S_1, \quad x_2 \in S_2. \end{aligned}$$

La funzione obiettivo può essere interpretata come un *profitto dipendente da due variabili decisionali*, x_1 e x_2, che devono appartenere ai rispettivi insiemi ammissibili S_1 e S_2.

Anche se i vincoli su x_1 e x_2 sono separabili, **non è possibile decomporre il problema complessivo**, poiché le due decisioni interagiscono tramite le funzioni di profitto $\pi_1(x_1; x_2)$ e $\pi_2(x_1; x_2)$. Risolvendo il problema si ottengono decisioni ottime x_1^* e x_2^* e il profitto totale ottimo $\pi_{1+2}^* = \pi_1(x_1^*, x_2^*) + \pi_2(x_1^*, x_2^*)$.

Così facendo si assume un **decisore unico** che prende entrambe le decisioni, oppure una coppia di decisori cooperativi che scelgono x_1 e x_2 condividendo l'obiettivo di massimizzare la somma dei profitti.

Caso non cooperativo Nel caso realistico di **due decisori non cooperativi** con profitti $\pi_1(x_1, x_2)$ e $\pi_2(x_1, x_2)$,

$$\text{Decisore 1: } \begin{array}{c}\max \pi_1(x_1, x_2) \\ \text{s.t. } x_1 \in S_1\end{array} \qquad \text{Decisore 2: } \begin{array}{c}\max \pi_2(x_1, x_2) \\ \text{s.t. } x_2 \in S_2\end{array}$$

Questi due problemi, scritti così, non hanno senso: quale valore di x_2 usare nell'altro problema? E viceversa. È necessario chiarire come i due decisori muovono.

1. Una possibilità è che i decisori agiscano in **modo sequenziale**. Ad esempio, il decisore 1 sceglie $x_1 \in S_1$ prima che il decisore 2 scelga $x_2 \in S_2$. In tal caso, il decisore 1 è il leader e il decisore 2 è il follower. Nel prendere la decisione, il decisore 1 può cercare di anticipare la reazione del decisore 2 a ciascun valore possibile di x_1.
2. Un'altra possibilità è che le due decisioni siano prese **simultaneamente**. In tal caso, servono strumenti concettuali per capire quali decisioni aspettarsi.

Equilibrio e perdita di performance La teoria dei giochi mira a fornire una previsione sensata di una **soluzione di equilibrio** (x_1^e, x_2^e), che dipende dalle ipotesi sulla struttura del gioco. Qualunque equilibrio si ottenga, esso non può produrre un profitto totale maggiore di π_{1+2}^*, poiché necessariamente vale

$$\pi_{1+2}^e = \pi_1(x_1^e, x_2^e) + \pi_2(x_1^e, x_2^e) \leq \pi_1(x_1^*, x_2^*) + \pi_2(x_1^*, x_2^*) = \pi_{1+2}^*.$$

Se questa disuguaglianza fosse violata, (x_1^*, x_2^*) non sarebbe soluzione ottima del problema iniziale. Questo significa che, **decentralizzando le decisioni, il sistema complessivo è verosimilmente incapace di raggiungere la performance ottima globale**.

FINIRE CAP 6 – SLIDE 54/90

7 CAP 7 – Il principio della programmazione dinamica

Recap La **programmazione dinamica (DP)** è un **principio generale e flessibile** per la risoluzione di **problemi decisionali dinamici multistadio**, basato sulla **decomposizione** del problema complessivo in una sequenza di problemi più semplici a **singolo stadio**. Nei problemi stocastici, a differenza dei problemi multiperiodo deterministici, le decisioni non vengono fissate interamente all'istante iniziale, ma si adattano nel tempo alle realizzazioni osservate dei fattori di rischio, rendendo necessario un approccio **closed-loop** e l'uso di **strategie non anticipative**.

L'applicazione della DP richiede una struttura **markoviana** del sistema, in cui lo stato s_t riassume l'informazione rilevante del passato e consente di descrivere l'evoluzione attraverso un'equazione di transizione del tipo $s_{t+1} = g_{t+1}(s_t, x_t, \xi_{t+1})$. Le decisioni x_t vengono prese dopo aver osservato lo stato corrente, mentre gli input esogeni si realizzano nell'intervallo temporale successivo e influenzano lo stato futuro. In questo contesto, lo stato, le decisioni e i fattori di rischio possono essere discreti o continui e l'orizzonte temporale può essere finito o infinito.

I problemi di programmazione dinamica sono tipicamente formulati tramite una **funzione obiettivo additiva nel tempo**, spesso con l'introduzione di un **fattore di sconto** e di un **contributo terminale**, la cui valutazione può richiedere un ragionamento *ad hoc*. Quando i contributi immediati dipendono da variabili aleatorie, si lavora con valori attesi condizionati che, in pratica, possono essere stimati solo tramite campionamento o simulazione, come avviene nel **reinforcement learning**.

Il **problema del cammino minimo** su una rete diretta aciclica rappresenta un esempio deterministico fondamentale per introdurre la **value function** e il principio di **programmazione dinamica all'indietro**. Associando a ciascun nodo un valore pari al costo minimo per raggiungere il nodo terminale, emerge una proprietà di **annidamento** dei sottocammini ottimi e si ottiene una ricorsione sui valori che

parte dalla condizione terminale. Questo esempio chiarisce il ruolo del valore dello stato successivo nella scelta locale delle decisioni.

Nel **lot-sizing** dinamico a singolo prodotto, l'inventario costituisce una **variabile di stato**, la quantità ordinata una **variabile di controllo** e la domanda agisce come input esogeno deterministico o come fattore di rischio stocastico. In presenza di domanda incerta, modelli diversi derivano da ipotesi alternative sul comportamento dei clienti, distinguendo tra *vendite perse* e *backlog*, con effetti diretti sulla dinamica e sulla struttura dei costi.

La formalizzazione generale dei problemi decisionali dinamici conduce all'**equazione di Bellman**, che esprime una condizione di ottimalità ricorsiva bilanciando **contributo immediato** e **valore atteso dello stato successivo**. Da essa si ricava una **politica ottima in feedback** della forma $x_t^\star = \mu_t^\star(s_t)$. Nei problemi a orizzonte finito la soluzione si ottiene naturalmente tramite una procedura **backward**, mentre nei problemi a orizzonte infinito la value function è caratterizzata come **punto fisso** di un opportuno operatore.

Che cos'è la programmazione dinamica? La **programmazione dinamica (Dynamic Programming, DP)** non è un algoritmo come l'algoritmo del simplesso per la programmazione lineare. È piuttosto un **principio notevolmente generale e flessibile** che può essere utilizzato per progettare una vasta gamma di algoritmi di ottimizzazione.

L'**idea centrale** si basa sulla **decomposizione di un problema decisionale dinamico multistadio** in una **sequenza di problemi più semplici a singolo stadio**.

Problemi multistadio e multiperiodo Un problema decisionale multistadio non deve essere confuso con un problema *multiperiodo*. Si consideri un problema a orizzonte finito in cui si devono selezionare T decisioni x_t, applicate in una sequenza di istanti temporali $t = 0, 1, 2, \ldots, T-1$. La domanda fondamentale è: *quando vengono prese queste decisioni*?

In un problema statico multiperiodo, tutte le decisioni vengono prese al tempo $t = 0$, come è naturale in un problema deterministico (**approccio open-loop**). In un problema stocastico, invece, si osserva nel tempo una traiettoria campionaria dei fattori di rischio rilevanti e si adattano le decisioni lungo il percorso.

Ciò che serve è una **strategia**, ovvero una **regola per prendere decisioni dopo aver osservato le realizzazioni casuali** (**approccio closed-loop**).

Ambito di applicazione della programmazione dinamica La programmazione dinamica non è un approccio universale, poiché la sua applicazione richiede una struttura specifica nel modello del sistema, che deve essere **markoviana**.

Nonostante ciò, la programmazione dinamica può essere applicata a una vasta gamma di problemi

- **Problemi continui e discreti**, dove la natura discreta o continua può riguardare: variabili di stato, variabili decisionali e rappresentazione del tempo.
- **Problemi deterministici e stocastici**.
- **Problemi a orizzonte finito e infinito**.

All'interno di questa ampia collezione di problemi, può essere necessario risolvere problemi a *dimensione finita o infinita*. Per alcune strutture, la programmazione dinamica rappresenta **l'unico approccio risolutivo praticabile**.

Problemi decisionali dinamici Si considerano modelli a tempo discreto ed è necessario chiarire la distinzione tra **istanti temporali** e **intervalli di tempo**.

Le *decisioni* vengono prese agli *istanti* temporali, ma possono essere *implementate* lungo un *intervallo* di tempo, non necessariamente in modo istantaneo. Inoltre, le decisioni devono essere prese prima di osservare la realizzazione dei fattori di rischio, che si manifestano nell'intervallo temporale successivo.

Si adotta la seguente **convenzione**

- Gli **istanti temporali** sono indicizzati da $t = 0, 1, 2, \ldots$; in tali istanti si osserva lo stato del sistema e si prende una decisione.
- L'**intervallo di tempo** t è l'intervallo tra gli istanti $t-1$ e t; dopo aver applicato la decisione presa all'istante $t-1$, il sistema evolve durante l'intervallo successivo e raggiunge un nuovo stato all'istante t. Durante tale intervallo può realizzarsi un input esterno, eventualmente casuale, che influenza la transizione di stato.

Orizzonte finito e infinito Un problema può avere un **orizzonte finito**, definito sugli istanti $t = 0, 1, \ldots, T$, oppure un **orizzonte infinito**, definito sugli istanti $t = 0, 1, \ldots$. Il primo istante temporale è $t = 0$, mentre il primo intervallo temporale è indicizzato da $t = 1$. In un problema a orizzonte finito, vi sono T intervalli di tempo e $T + 1$ istanti temporali.

La dinamica del sistema è rappresentata da un'**equazione di transizione di stato** del tipo

$$s_{t+1} = g_{t+1}(s_t, x_t, \xi_{t+1}),$$

dove

- g_{t+1} è la **funzione di transizione** sul periodo $t + 1$.
- s_t è il **vettore delle variabili di stato** all'istante t, che riassumono tutta l'informazione rilevante del passato; s_0 è lo **stato iniziale**, tipicamente noto, mentre s_T è lo stato terminale.
- x_t è il **vettore delle variabili decisionali** all'istante t, dette anche variabili di controllo.
- ξ_{t+1} è un **fattore esogeno** che si realizza durante l'intervallo $t + 1$.

Le decisioni x_t vengono prese all'istante di tempo t, dopo aver osservato lo stato s_t; lo stato successivo s_{t+1} dipende dalla realizzazione della variabile aleatoria ξ_{t+1}.

Variabili di stato e processi Si distinguono diverse tipologie di **variabili di stato**

- **Variabili di stato fisiche**, che descrivono risorse fisiche o finanziarie e sono direttamente influenzate dalle decisioni.
- **Variabili di stato informative**, come il prezzo di un'azione finanziaria, non influenzate dall'azione di un singolo decisore.
- **Variabili di stato di credenza**, rilevanti in problemi con incertezza sui parametri del modello.

Si definiscono: il **processo dei dati** $(\xi_t)_{t\geq 1}$, il **processo decisionale** $(x_t)_{t\geq 0}$ e il **processo di stato** $(s_t)_{t\geq 0}$, che possono essere deterministici o stocastici.

Informazione e non anticipatività Nei problemi decisionali dinamici sotto incertezza, due aspetti sono cruciali

1. Come modellare la *disponibilità dell'informazione*.
2. Come sono *correlati i processi* dei dati, dello stato e delle decisioni.

Si introduce la notazione $x_{[t]} := (x_0, x_1, \ldots, x_t)$, $\xi_{[t]} := (\xi_1, \xi_2, \ldots, \xi_t)$, utile per rappresentare la storia osservata fino all'istante di tempo t.

La decisione x_t può dipendere solo dalle realizzazioni dei fattori di rischio fino all'istante t incluso. Una politica decisionale implementabile è detta **non anticipativa**.

Dipendenza temporale dei fattori di rischio I vettori casuali del processo dei dati possono presentare diversi livelli di dipendenza: *indipendenza interstadio*, *dipendenza markoviana* e *dipendenza dall'intera storia osservata*.

È inoltre necessario specificare se la distribuzione dei fattori di rischio dipende dal tempo o dallo stato e dalle decisioni.

L'**obiettivo** è **ottimizzare una funzione obiettivo additiva nel tempo**, pari alla somma dei costi o dei ricavi generati in ciascun istante.

7.1 Problemi a orizzonte finito scontati

Definizione Un problema stocastico a orizzonte finito T può essere formulato

$$\text{opt } \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \gamma^t f_t(s_t, x_t) + \gamma^T F_T(s_T) \right],$$

dove opt indica una minimizzazione o massimizzazione. La notazione $\mathbb{E}_0[\cdot]$ viene utilizzata per indicare che l'aspettativa è valutata all'istante di tempo $t = 0$; si tratta quindi di un'aspettativa incondizionata, poiché non è stata ancora osservata alcuna realizzazione dei fattori di rischio (ad eccezione di quelle che hanno condotto allo stato iniziale s_0).

La **funzione obiettivo** include

- Il **contributo immediato** $f_t(s_t, x_t)$ in cui incorriamo prendendo la decisione x_t quando siamo nello stato s_t.
- Il **contributo terminale** $F_T(s_T)$, che dipende solamente dallo stato terminale s_T.

Fattore di sconto Il fattore di sconto $\gamma \in (0, 1)$ è tipico nelle applicazioni finanziarie, anche se non è strettamente necessario in un problema a orizzonte finito perchè la funzione obiettivo è ben definita. Individuare un modo appropriato per associare un valore allo stato terminale può richiedere un ragionamento *ad hoc*.

Contributi stocastici e apprendimento Il contributo immediato può dipendere dalla realizzazione del fattore di rischio $h_t(s_t, x_t, \xi_{t+1})$. In tal caso, si può definire un'attesa condizionata

$$f_t(s_t, x_t) = \mathbb{E}_t \left[h_t(s_t, x_t, \xi_{t+1}) \right].$$

In pratica, il calcolo dell'attesa può essere difficile o impossibile. In tali casi, si dispone solo di campioni osservabili tramite simulazione Monte Carlo o sperimentazione online, come avviene nel **reinforcement learning**.

7.2 Problemi a orizzonte infinito

Problemi scontati a orizzonte infinito Un **problema di programmazione dinamica scontato a orizzonte infinito** ha la forma

$$\text{opt } \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \gamma^t f(s_t, x_t) \right],$$

dove la sommatoria è ben definita se tutti i contributi sono positivi e limitati e si utilizza un fattore di sconto appropriato $\gamma < 1$.

La programmazione dinamica scontata è naturale nelle applicazioni di business ed economia. Talvolta il fattore di sconto non ha una reale giustificazione economica ed è utilizzato solo come espediente per rendere trattabile un problema a orizzonte infinito.

In alcune applicazioni ingegneristiche si preferisce considerare una media nel tempo

$$\text{opt } \lim_{T \to \infty} \mathbb{E}_0 \left[\frac{1}{T} \sum_{t=0}^{T-1} f(s_t, x_t) \right].$$

Questo conduce a formulazioni di programmazione dinamica basate sul contributo medio per stadio.

Esistono anche problemi con orizzonte non definito, nei quali il processo decisionale termina quando viene raggiunto uno stato obiettivo.

Politiche decisionali Ad ogni istante temporale occorre selezionare una decisione x_t. Si può introdurre un vincolo astratto del tipo: $x_t \in X(s_t)$, che impone che la decisione al tempo t appartenga a un insieme ammissibile X, eventualmente dipendente dallo stato corrente s_t.

Caso deterministico e stocastico In un contesto **deterministico**, questo vincolo sarebbe sufficiente, poiché al tempo $t = 0$ si dovrebbe individuare una sequenza di decisioni $(x_t)_{t\geq 0}$ da implementare nel tempo. Nel caso **stocastico**, tuttavia, entra in gioco un vincolo fondamentale. Le decisioni devono essere **non anticipative**, cioè non possono dipendere da informazioni future non ancora osservate.

Una *politica decisionale chiusa* e *non anticipativa* è naturalmente espressa nella forma

$$x_t = \mu_t(s_t) \in X(s_t),$$

dove $\mu_t(\cdot)$ è una funzione che associa allo stato al tempo t una decisione ammissibile.

Una politica può essere vista come una **sequenza di funzioni**

$$\mu \equiv \big(\mu_0(\cdot), \mu_1(\cdot), \ldots, \mu_{T-1}(\cdot)\big),$$

una per ciascun istante temporale in cui è richiesta una decisione.

Nel caso di problemi a *orizzonte infinito*, si cercano politiche **stazionarie**, caratterizzate dalla stessa funzione $\mu(\cdot)$ in ogni stadio.

Formulazione del problema in termini di politiche Una formulazione più precisa del problema a orizzonte finito è

$$\mathrm{opt}_{\mu\in\mathcal{M}}\, \mathbb{E}_0 \left[\sum_{t=0}^{T-1} \gamma^t f_t(s_t, \mu_t(s_t)) + \gamma^T F_T(s_T)\right],$$

dove $\mathcal{M}$ è l'insieme delle politiche ammissibili, tali che $x_t = \mu_t(s_t) \in X(s_t)$ per ogni istante temporale.

Sono state introdotte **politiche deterministiche**. Esistono tuttavia casi in cui politiche randomizzate possono essere necessarie o utili.

Le **politiche randomizzate** possono essere richieste per gestire vincoli probabilistici sugli stati: $\mathbb{P}\{s_t \in G\} \geq 1 - \alpha$, dove α è una **probabilità accettabile di violazione del vincolo**. Le politiche randomizzate risultano inoltre necessarie nel contesto del trade-off tra esplorazione e exploitation nel reinforcement learning.

Un esempio: il lot-sizing dinamico a singolo prodotto Si consideri il problema di lot-sizing non capacitated a singolo prodotto

$$\begin{aligned} \min\ & \sum_{t=0}^{T-1} \big[c x_t + \phi \cdot \delta(x_t)\big] + \sum_{t=1}^{T} h I_t \\ \text{s.t. } & I_{t+1} = I_t + x_t - d_{t+1}, \qquad t = 0, 1, \ldots, T-1, \\ & x_t \geq 0,\ I_t \geq 0. \end{aligned}$$

L'**obiettivo** è soddisfare la domanda al costo minimo, che comprende: un *costo di mantenimento a magazzino*, con tasso h per unità e per unità di tempo e un *costo di ordinazione*, composto da una parte variabile con coefficiente c e da un costo fisso ϕ, sostenuto solo quando si ordina una quantità positiva. Per rappresentare il costo fisso si introduce la funzione

$$\delta(x) = \begin{cases} 1 & \text{se } x > 0, \\ 0 & \text{se } x = 0. \end{cases}$$

Il **livello di inventario** I_t è una variabile di stato, la quantità ordinata x_t è una variabile di controllo, e la domanda d_t può essere interpretata come input esogeno nel caso deterministico o come fattore di rischio nel caso stocastico.

Domanda incerta e modelli alternativi Come si dovrebbe adattare il modello nel caso in cui si consideri l'incertezza della domanda? Esistono due modelli di base, che dipendono da assunzioni specifiche sul comportamento dei clienti

1. Nel caso di vendite perse, l'equazione di transizione diventa: $I_{t+1} = \max\{0, I_t + x_t - d_{t+1}\}$. Si introduce inoltre una penalità q per ogni unità di domanda insoddisfatta, aggiungendo il termine: $\sum_{t=1}^{T} q \max\{0, d_t - (I_{t-1} + x_{t-1})\}$.
2. Se i clienti sono pazienti, la domanda può essere accumulata come backlog, con una penalità b, con $b > h$. Una prima opzione è mantenere una singola variabile di stato I_t, rilassandone la non negatività e interpretando valori negativi come backlog. Una seconda opzione è introdurre due variabili di stato: *inventario disponibile* $O_t \geq 0$ e *backlog* $B_t \geq 0$. Il costo diventa

$$\sum_{t=1}^{T} (hO_t + bB_t),$$

e le equazioni di transizione sono: $O_{t+1} = \max\{0, O_t - B_t + x_t - d_{t+1}\}$, $B_{t+1} = \max\{0, -O_t + B_t - x_t + d_{t+1}\}$.

Ulteriori problematiche In applicazioni reali possono sorgere ulteriori difficoltà: valutazione dello **stato terminale**, stati *parzialment* osservabili, domanda censurata, **ritardi** dovuti a lead time di consegna e *incertezza* dipendente dallo stato e dalle decisioni.

7.3 Uno sguardo al principio di programmazione dinamica: il problema del cammino minimo

Rete diretta aciclica Si consideri un **problema deterministico**: il cammino minimo su una rete diretta aciclica. Una **rete diretta** è costituita da un insieme di nodi $N = \{0, 1, 2, \ldots, N\}$ e da un insieme di archi A. Un **arco** è una coppia ordinata $(i, j) \in A$, con $i, j \in N$, ed è associato a un costo $c_{ij} > 0$. Si assume che la rete sia *aciclica*.

Per un **nodo** $i \in N$ si definiscono, rispettivamente,

$$S_i = \{j \in N \mid (i, j) \in A\}, \qquad B_i = \{j \in N \mid (j, i) \in A\},$$

il set di **successori** e **predecessori** del nodo i. Un **cammino diretto** da i a j è una sequenza di nodi $(k_1, k_2, \ldots, k_m)$ con $k_1 = i$, $k_m = j$, e archi consecutivi in A. La **lunghezza del cammino** è la somma dei costi sugli archi.

Ogni *nodo* è uno *stato* e ogni *arco* una *transizione* possibile. In ogni stato occorre scegliere una transizione in modo da raggiungere lo stato terminale al costo minimo.

Approccio greedy Una **regola greedy** consisterebbe nel muoversi verso il nodo più vicino

$$\min_{j \in S_i} c_{ij}.$$

Tale approccio non è in generale ottimale. Tuttavia, l'**idea** di risolvere una sequenza di semplici problemi a singolo stadio presenta un certo fascino. Può quindi essere utile affinare l'**obiettivo** introducendo una misura della bontà dello stato successivo. Si introduce il valore V_j associato a ciascun nodo successore, risolvendo in i il problema:

$$\min_{j \in S_i} (c_{ij} + V_j).$$

Definizione del valore ottimo *Come si può introdurre un termine aggiuntivo di questo tipo?* Il punto di partenza consiste nel trovare una caratterizzazione della soluzione ottima. Sia V_i la **lunghezza del cammino minimo** dal nodo $i \in N$ al nodo terminale N, indicato come $i \xrightarrow{*} N$: $V_i := L(i \xrightarrow{*} N)$.

Supponiamo inoltre che, dato un cammino ottimo da i a N, un nodo j appartenga a tale cammino. Allora è immediato osservare che deve valere la seguente proprietà

$$j \xrightarrow{*} N \text{ è un sottocammino di } i \xrightarrow{*} N.$$

Per comprenderne il motivo, consideriamo la decomposizione del cammino $i \xrightarrow{*} N$ in due sottocammini: $\mathcal{P}_{i\to j}$ dal nodo i al nodo j, e $\mathcal{P}_{j\to N}$ dal nodo j al nodo N. La lunghezza del cammino $i \xrightarrow{*} N$ è la somma delle lunghezze dei due sottocammini: $V_i = L\left(i \xrightarrow{*} N\right) = L(\mathcal{P}_{i\to j}) + L(\mathcal{P}_{j\to N})$.

Si noti che il secondo sottocammino non è influenzato dal modo in cui si va da i a j. Il sistema è **markoviano**, nel senso che il modo in cui si raggiunge il nodo j non ha alcuna influenza sulla lunghezza di qualunque cammino *futuro* che parte dal nodo j.

L'affermazione equivale a dire che il sottocammino $\mathcal{P}_{j\to N}$ nella decomposizione precedente è **ottimo**, nel senso che $L(\mathcal{P}_{j\to N}) = L\left(j \xrightarrow{*} N\right)$. Per **dimostrarlo**, si suppone che $\mathcal{P}_{j\to N}$ non sia un cammino ottimo da j a N, cioè che $L(\mathcal{P}_{j\to N}) > L\left(j \xrightarrow{*} N\right)$. Allora potremmo migliorare il lato destro della decomposizione considerando il cammino costituito dallo stesso cammino iniziale $\mathcal{P}_{i\to j}$, seguito da un cammino ottimo $j \xrightarrow{*} N$: $L(\mathcal{P}_{i\to j}) + L\left(j \xrightarrow{*} N\right) < L(\mathcal{P}_{i\to j}) + L(\mathcal{P}_{j\to N}) = L\left(i \xrightarrow{*} N\right) = V_i$, che è una contraddizione, poiché abbiamo assunto che V_i sia la lunghezza di un cammino minimo.

I cammini minimi godono quindi di una sorta di **proprietà di annidamento**, e questo risultato suggerisce una decomposizione ricorsiva del problema complessivo di determinare il cammino minimo $0 \xrightarrow{*} N$.

Se si conoscessero i valori V_j per ciascun nodo $j \in \mathcal{S}_0$, si potrebbe effettuare la prima decisione risolvendo il problema a singolo stadio

$$V_0 = \min_{j\in\mathcal{S}_0}\left(c_{0j} + V_j\right).$$

Ricorsione di Bellman L'ultima equazione può essere applicata a qualunque nodo, e non soltanto a quello iniziale. Si ottiene così una **programmazione dinamica all'indietro**, che parte dalla **condizione terminale** $V_N = 0$ e procede risolvendo un'**equazione funzionale ricorsiva**

$$V_i = \min_{j\in S_i}\left(c_{ij} + V_j\right), \qquad \forall i \in N,$$

con condizione terminale $V_N = 0$. L'equazione viene risolta etichettando i nodi a partire da quello terminale, seguendo un **ordinamento topologico**. L'equazione precedente definisce una **value function** $V(\cdot) : N \to \mathbb{R}$ che associa a ciascuno stato il suo valore ottimo.

Cammini minimi su reti strutturate Il problema del cammino minimo considerato nella sezione precedente riguarda una rete non strutturata. Un caso più strutturato si verifica quando i nodi della rete corrispondono a stati di un sistema dinamico che evolve nel tempo. Questo tipo di grafo corrisponde a un **problema decisionale sequenziale deterministico** con un **insieme finito di stati**, che può derivare da una discretizzazione di uno spazio degli stati continuo.

Se lo stato terminale s_T è fissato, non ha senso associare un valore terminale ad esso. Se lo stato terminale è libero, può invece avere senso associare un contributo terminale $F_T(s_T)$. In tal caso si aggiunge uno strato temporale al grafo e si introduce un nodo terminale fittizio Ω.

7.4 Il principio di decomposizione della programmazione dinamica – Equazione di Bellman

Decomposizione in problemi a singolo stadio Il problema del cammino minimo suggerisce che un problema decisionale multistadio può essere decomposto in una sequenza di problemi più semplici a singolo

stadio.

In un **problema deterministico**, si cerca una sequenza di vettori $(x_0, x_1, \ldots, x_{T-1})$ per ottimizzare una misura di prestazione opt $H(x_0, x_1, \ldots, x_{T-1}; s_0, s_1, \ldots, s_T)$, soggetta a un insieme di vincoli su stati e decisioni.

Nel **caso stocastico**, il problema è opt $\mathbb{E}[H(x_0, x_1, \ldots, x_{T-1}; s_0, s_1, \ldots, s_T)]$, dove l'attesa è presa rispetto a una sequenza di variabili aleatorie $(\xi_1, \ldots, \xi_T)$. Questa notazione nasconde la vera natura multistadio del problema, poiché ora si dovrebbero trovare una sequenza di funzioni.

Infatti, a parte la decisione iniziale x_0 da prendere *qui e ora*, i vettori decisionali possono essere funzioni delle decisioni passate e delle realizzazioni osservate dei fattori di rischio

$$x_t = \mu_t\big(x_{[t-1]}, \xi_{[t]}\big), \qquad t = 1, \ldots, T-1.$$

Se si individua un insieme adatto di **variabili di stato**, si può semplificare la dipendenza dalla storia e cercare funzioni più semplici che mappano lo stato corrente nella decisione ottima, cioè $x_t = \mu_t(s_t)$.

Additività della funzione obiettivo Oltre alla struttura **markoviana** della dinamica, una decomposizione *pulita* è possibile se si assume che la funzione obiettivo abbia forma additiva

$$\mathbb{E}_0\left[\sum_{t=0}^{T-1} \gamma^t f_t(s_t, x_t) + \gamma^T F_T(s_T)\right].$$

Una **regola decisionale rapida** e approssimativa è: *quando si è nello stato s_t, risolvere il problema miopico* $\text{opt}_{x_t \in X(s_t)} f_t(s_t, x_t)$, dove $X(s_t)$ è l'insieme delle decisioni ammissibili nello stato s_t.

È già noto che tale approccio **greedy** non è atteso funzionare bene in generale: occorre bilanciare obiettivi di breve e lungo periodo introducendo una **value function** $V_t(\cdot)$.

L'**idea fondamentale** della programmazione dinamica è che si può trovare una value function tale da ottenere la prestazione ottima. Il valore $V_t(s)$ dovrebbe essere il costo/ricavo atteso ottenuto applicando una politica ottima dal tempo t in avanti, partendo dallo stato s.

Problema a singolo stadio parametrizzato dallo stato Formalmente, nello stato s_t al tempo t si deve risolvere

$$V_t(s_t) = \text{opt}_{x_t \in X(s_t)} \Big\{ f_t(s_t, x_t) + \gamma\, \mathbb{E}\big[V_{t+1}\big(g_{t+1}(s_t, x_t, \xi_{t+1})\big) \;\big|\; s_t, x_t\big] \Big\}.$$

Questa equazione funzionale ricorsiva è detta **equazione di Bellman**.

La decisione ottima $x_t^\star$ si ottiene risolvendo un problema di ottimizzazione parametrizzato dallo stato corrente s_t, usando la conoscenza della value function $V_{t+1}(\cdot)$. Nel caso di uno *spazio degli stati finito* e piccolo, la politica ottima può essere rappresentata in **forma tabellare**. In generale, la politica è implicita nella sequenza delle funzioni valore. In ogni caso, concettualmente si trova una politica ottima in forma di feedback: $x_t^\star = \mu_t^\star(s_t) \in X(s_t)$. Raccogliendo tutte le funzioni $\mu_t^\star(\cdot)$ si ottiene la politica complessiva: $\mu^\star \equiv \big(\mu_0^\star(\cdot), \mu_1^\star(\cdot), \ldots, \mu_{T-1}^\star(\cdot)\big)$.

Quando si considera una singola funzione $\mu^\star(\cdot)$ indipendente dal tempo si parla di **politica stazionaria**, adatta ai problemi a orizzonte infinito. Talvolta ci si accontenta di una politica subottima, ad esempio ottenuta da un'approssimazione della value function ottima.

Teorema: principio di ottimalità della programmazione dinamica Si consideri una politica ottima $\big(\mu_0^\star(\cdot), \mu_1^\star(\cdot), \ldots, \mu_{T-1}^\star(\cdot)\big)$ per il **problema multistadio**

$$\text{opt}\;\; \mathbb{E}_0\left[\sum_{t=0}^{T-1} \gamma^t f_t(s_t, x_t) + \gamma^T F_T(s_T)\right].$$

Si assuma che al tempo τ ci si trovi nello stato s_τ e si consideri il problema di coda

$$\text{opt}\;\; \mathbb{E}_\tau\left[\sum_{t=\tau}^{T-1} \gamma^{t-\tau} f_t(s_t, x_t) + \gamma^{T-\tau} F_T(s_T)\right].$$

Allora la politica troncata $\big(\mu_\tau^\star(\cdot), \mu_{\tau+1}^\star(\cdot), \dots, \mu_{T-1}^\star(\cdot)\big)$ è ottima per il problema di coda.

Le **dimostrazioni** si basano sull'induzione matematica e possono essere piuttosto complicate quando si considerano questioni matematiche sottili.

7.5 DP stocastica per orizzonti finiti

Risoluzione all'indietro L'**equazione di Bellman** è un'*equazione di ottimalità* e richiede di determinare la value function per ogni istante temporale. Il processo naturale procede all'indietro nel tempo, partendo dalla condizione terminale: $V_T(s_T) = F_T(s_T) \quad \forall s_T$.

All'ultimo istante decisionale, $t = T-1$, si risolve per ogni possibile stato s_{T-1}

$$V_{T-1}(s_{T-1}) = \text{opt}_{x_{T-1}\in X(s_{T-1})}\Big\{ f_{T-1}(s_{T-1}, x_{T-1}) + \gamma\, \mathbb{E}\big[V_T\big(g_T(s_{T-1}, x_{T-1}, \xi_T)\big) \;\big|\; s_{T-1}, x_{T-1}\big]\Big\}.$$

Questo è un *problema statico ma non miopico*, poiché $V_T(\cdot)$ incorpora l'effetto della decisione x_{T-1} sullo stato terminale.

Risolvendo il problema precedente per ogni stato s_{T-1} si costruisce la value function $V_{T-1}(\cdot)$. Poi, svolgendo la ricorsione all'indietro nel tempo, si ottengono $V_{T-2}(s_{T-2})$ e così via, fino a $V_1(s_1)$. Infine, dato lo stato iniziale s_0, la prima decisione ottima si trova risolvendo

$$V_0(s_0) = \text{opt}_{x_0\in X(s_0)}\Big\{ f_0(s_0, x_0) + \gamma\, \mathbb{E}\big[V_1\big(g_1(s_0, x_0, \xi_1)\big) \;\big|\; s_0, x_0\big]\Big\}.$$

Uso delle funzioni valore: caso deterministico *Come sfruttare la conoscenza delle funzioni valore $V_t(\cdot)$?* In un contesto **deterministico**, si può trovare la sequenza di decisioni ottime $x_t^\star$ risolvendo una sequenza di problemi a singolo stadio e aggiornando lo stato secondo le decisioni applicate.

In un contesto **stocastico**, si può eseguire una simulazione ***Monte Carlo*** come segue. Dato lo stato iniziale s_0 e la value function $V_1(\cdot)$, si risolve il problema del primo stadio e si trova $x_0^\star$; quindi si campiona ξ_1 e si genera lo stato successivo $s_1 = g_1(s_0, x_0^\star, \xi_1)$. Ripetendo lo stesso ragionamento, dato lo stato s_t e la value function $V_{t+1}(\cdot)$, si risolve il problema del tempo t ottenendo $x_t^\star$, e si prosegue fino a generare l'ultima decisione $x_{T-1}^\star$ e lo stato terminale s_T.

DP stocastica per orizzonti infiniti La forma ricorsiva deve essere adattata per un problema a orizzonte infinito del tipo

$$\text{opt}\;\; \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t f(s_t, x_t)\right],$$

dove si assume che i costi immediati siano limitati e $\gamma < 1$, così che la serie converga a un valore finito.

Si elimina l'indice t dal contributo immediato e dalla funzione di transizione: $s_{t+1} = g(s_t, x_t, \xi_{t+1})$.

L'equazione funzionale diventa

$$V(s) = \text{opt}_{x\in X(s)}\Big\{ f(s, x) + \gamma\, \mathbb{E}\big[V\big(g(s, x, \xi)\big)\big]\Big\},$$

dove $X(s)$ è l'insieme delle decisioni ammissibili nello stato s.

Ora la value function è definita come **punto fisso** di un operatore potenzialmente complicato, poiché $V(s)$ compare su entrambi i lati dell'equazione. È necessario un *metodo iterativo* per risolvere tale equazione.

8 CAP 8 – Implementazione della programmazione dinamica

Recap L'**allocazione di risorse tramite programmazione dinamica** viene affrontata in diversi contesti, mostrando come il principio DP possa essere implementato operativamente in problemi discreti, continui e stocastici. L'attenzione è posta sulla costruzione e sull'uso della **value function** come strumento computazionale, mettendo in evidenza il ruolo della variabile di stato e delle decisioni di controllo.

Nel contesto di **allocazione discreta**, il problema dello **zaino** fornisce un modello elementare ma rappresentativo. Un problema statico di selezione binaria viene riformulato come processo sequenziale introducendo uno stadio fittizio associato agli oggetti, con il **budget residuo** come stato. In questo caso la value function è definita su un insieme finito e può essere **tabulata**, portando a una ricorsione di Bellman semplice e a un'implementazione numerica diretta.

L'**allocazione continua del budget** estende il problema a decisioni reali e a funzioni di profitto **crescente e concave**. In presenza di soluzioni interne, il problema ammette una caratterizzazione analitica tramite condizioni di ottimalità. Tuttavia, la stessa struttura può essere descritta in termini di programmazione dinamica, con una value function definita su uno spazio continuo.

Nel caso continuo, la value function è un oggetto **infinito-dimensionale** e non può essere memorizzata esplicitamente. La risoluzione numerica richiede quindi **discretizzazione dello stato** e tecniche di **approssimazione**. L'uso di **spline cubiche** consente di interpolare i valori fuori griglia evitando le oscillazioni tipiche dell'interpolazione polinomiale globale e preservando, almeno approssimativamente, proprietà qualitative della funzione valore.

La programmazione dinamica viene applicata anche al **controllo stocastico delle scorte**, in cui la domanda è modellata come variabile aleatoria discreta. In questo contesto il costo immediato dipende dalla realizzazione futura del fattore di rischio e la ricorsione DP include un termine in **aspettativa**. La modellazione richiede la specificazione della distribuzione della domanda, dello stato iniziale e dell'orizzonte temporale.

Nel **caso deterministico**, emergono proprietà strutturali forti che restringono l'insieme delle decisioni rilevanti. In particolare, la condizione di Wagner–Whitin consente di riformulare il problema di lot-sizing come un **cammino minimo** su una rete di dimensioni contenute, riducendo drasticamente la complessità computazionale.

In ambiente **stocastico**, tali proprietà non valgono più in generale, ma sotto ipotesi di convessità emergono politiche strutturate. In assenza di costi fissi si ottengono politiche di tipo **base-stock**, mentre l'introduzione di costi fissi conduce a politiche (s, S), caratterizzate da soglie di riordino e livelli obiettivo.

L'insieme di questi modelli evidenzia anche i **limiti della programmazione dinamica**, legati alla crescita dello spazio degli stati, alla difficoltà dei sottoproblemi di ottimizzazione, al costo del calcolo delle aspettative e alla complessità della modellazione delle transizioni quando il sistema dipende dalle decisioni di controllo.

8.1 Allocazione discreta di risorse: il problema dello zaino

Definizione, formulazione e motivazione DP Il problema dello **zaino** richiede di selezionare un sottoinsieme di oggetti di **valore totale massimo**, rispettando un vincolo di **capacità** (budget).

Per modellare la selezione di ciascun oggetto si introducono variabili decisionali **binarie**

$$x_k = \begin{cases} 1 & \text{se l'oggetto } k \text{ è selezionato,} \\ 0 & \text{altrimenti.} \end{cases}$$

Il problema può essere formulato come un problema di **programmazione lineare binaria pura**

$$\begin{aligned} \max \quad & \sum_{k=1}^{n} v_k x_k \\ \text{s.t.} \quad & \sum_{k=1}^{n} w_k x_k \leq B, \\ & x_k \in \{0,1\} \quad \forall k. \end{aligned}$$

Si assume un'allocazione discreta del budget, poiché la selezione di un'attività è una decisione **tutto-o-niente**. Il problema può essere risolto con algoritmi **branch-and-cut**, ma si adotta un approccio basato sul principio della **programmazione dinamica**.

Riformulazione sequenziale, stato e transizione Il problema non è intrinsecamente dinamico, ma può essere riformulato come un problema di **allocazione sequenziale di risorse** introducendo un indice discreto fittizio k.

Allo stadio iniziale si dispone del budget B, mentre allo stadio finale resta da valutare solo l'ultimo oggetto dato il budget residuo. La dipendenza dalle decisioni passate è interamente riassunta dal **budget residuo**. La variabile di stato allo stadio k è il budget disponibile s_k, mentre la decisione è la variabile binaria x_k. La transizione dello stato è

$$s_{k+1} = s_k - w_k x_k, \qquad s_1 = B.$$

value function e ricorsione di Bellman Si definisce la value function

$$V_k(s) = \text{profitto della selezione ottimale degli oggetti } \{k, \dots, n\} \text{ con budget residuo } s.$$

Nel caso **discreto** è possibile tabulare tutte le funzioni valore. La *ricorsione DP* è

$$V_k(s) = \begin{cases} V_{k+1}(s) & 0 \le s < w_k, \\ \max\{V_{k+1}(s), V_{k+1}(s - w_k) + v_k\} & w_k \le s \le B. \end{cases}$$

L'equazione esprime che, allo stadio k, viene considerato l'oggetto k. **Se** il suo peso w_k non è compatibile con il budget residuo s, cioè se $s < w_k$, l'oggetto non può essere selezionato. In questo caso la variabile di stato rimane invariata e la value function nello stato s coincide con quella dello stadio successivo, ossia $V_k(s) = V_{k+1}(s)$.

Se invece $s \ge w_k$, è necessario confrontare due alternative. La *prima* consiste nell'includere l'oggetto k nel sottoinsieme, ottenendo una ricompensa pari al suo valore v_k e allocando in modo ottimale il budget aggiornato $s - w_k$ agli oggetti successivi $k+1, \dots, n$. La *seconda* consiste nel non includere l'oggetto k, allocando l'intero budget residuo s in modo ottimale agli oggetti $k+1, \dots, n$. Poiché la decisione è binaria, il problema di ottimizzazione a singolo stadio risulta immediato. Per l'ultimo oggetto, corrispondente allo stadio $k = n$, la value function è definita come

$$V_n(s) = \begin{cases} 0 & \text{se } 0 \le s < w_n, \\ v_n & \text{se } w_n \le s \le B. \end{cases}$$

Implementazione MATLAB e complessità La funzione `DPKnapsack` memorizza la value function in una tabella indicizzata per stato e stadio e una tabella delle decisioni ottime.

La ricorsione è calcolata backward e la soluzione ottima è ricostruita forward. Assumendo dati interi, la complessità è $O(nB)$, quindi **pseudo-polinomiale**.

8.2 Allocazione continua del budget

Formulazione e soluzione analitica Si considera una versione **continua** del problema di allocazione delle risorse, in cui un budget complessivo B deve essere allocato tra un insieme di n attività.

L'allocazione all'attività k è rappresentata da una variabile decisionale continua $x_k \ge 0$, per $k = 1, \dots, n$. Il **contributo al profitto** dell'attività k dipende dall'allocazione di risorse tramite una funzione $f_k(\cdot)$ **crescente** e **concava**.

$$\begin{aligned} \max \quad & \sum_{k=1}^{n} f_k(x_k) \\ \text{s.t.} \quad & \sum_{k=1}^{n} x_k \le B, \\ & x_k \ge 0 \quad \forall k. \end{aligned}$$

Se le funzioni di profitto sono **concave**, ad esempio $f_k(x) = \sqrt{x}$ per $k = 1, \dots, n$, il problema risulta relativamente semplice da risolvere. Poiché le funzioni sono strettamente crescenti, il budget è interamente utilizzato. Assumendo una soluzione interna, la Lagrangiana conduce a un'allocazione uniforme $x_k^* = B/n$.

Soluzione interna e Lagrangiana Assumendo una **soluzione interna** $x_k^* > 0$, il problema diventa un *problema non lineare* con un solo vincolo di uguaglianza. Introducendo il moltiplicatore di Lagrange λ, la **lagrangiana** è

$$\mathcal{L}(x, \lambda) = \sum_{k=1}^{n} \sqrt{x_k} + \lambda \left(\sum_{k=1}^{n} x_k - B \right).$$

Le condizioni del primo ordine sono: $\frac{1}{2\sqrt{x_k}} + \lambda = 0, \quad k = 1, \ldots, n, \ \sum_{k=1}^{n} x_k - B = 0$, che implicano un'allocazione **uniforme**

$$x_k^* = \frac{B}{n}, \quad k = 1, \ldots, n.$$

In questo caso, la funzione obiettivo è **concava** e le condizioni di ottimalità sono **necessarie e sufficienti**.

Formulazione DP Il problema può essere riformulato in un quadro di **programmazione dinamica** introducendo un *indice discreto fittizio* $k = 1, \ldots, n$ associato alle attività. Sia $V_k(s)$ il **profitto ottimale** ottenibile allocando un budget residuo s alle attività $\{k, k+1, \ldots, n\}$. La **transizione di stato** è $s_{k+1} = s_k - x_k$, con condizione iniziale $s_1 = B$. Le **value funcitons** soddisfano

$$V_k(s_k) = \max_{0 \le x_k \le s_k} \{ f_k(x_k) + V_{k+1}(s_k - x_k) \}, \quad V_n(s_n) = \max_{0 \le x_n \le s_n} f_n(x_n) = f_n(s_n).$$

Il vincolo $s_k \ge 0$ è garantito imponendo il vincolo sulla decisione $0 \le x_k \le s_k$, poiché la transizione è deterministica.

Nel **caso continuo**, la value function $V_k(s)$ è un oggetto in uno spazio funzionale **infinito-dimensionale**. Poiché la valutazione è possibile solo su un insieme finito di stati, occorre usare **approssimazione** o **interpolazione** per stimare i valori fuori dalla griglia.

Interpolazione con spline cubiche in MATLAB È noto che l'interpolazione polinomiale può soffrire di **oscillazioni inaccettabili**. Una funzione **concava** e **monotona crescente** può essere approssimata da una funzione **non monotona**, con effetti negativi sulle procedure di ottimizzazione.

Un'alternativa standard consiste nell'uso di funzioni polinomiali a tratti di ordine più basso, in cui ogni tratto è associato a un sottointervallo della griglia. Una scelta comune sono le **spline cubiche**, ottenute in MATLAB tramite `spline` (costruzione) e `ppval` (valutazione in punti arbitrari, anche fuori griglia).

L'**errore** può essere significativo con griglie grossolane e migliora con griglie più fitte. Restano aperti temi come: la garanzia che una funzione monotona venga approssimata da una spline monotona, la scelta dei nodi per un buon compromesso tra costo computazionale ed errore, e la generalizzazione a dimensioni superiori.

Risoluzione numerica del problema continuo tramite DP Si usa una **spline cubica** per approssimare la value function del problema continuo. Si imposta una griglia uniforme su $[0, B]$, replicata per ciascuno stadio. La griglia contiene $m+1$ valori di stato, con *passo di discretizzazione* $\delta s = B/m$, cioè stati del tipo $j\,\delta s$, $j = 0, 1, \ldots, m$. Per ogni punto della griglia si risolve un sottoproblema del tipo

$$V_k(s_k) = \max_{0 \le x_k \le s_k} \{ f_k(x_k) + V_{k+1}(s_k - x_k) \},$$

tramite **ottimizzazione numerica**. I valori fuori dalla griglia sono stimati via spline cubiche. La condizione al bordo su $V_n(\cdot)$ è banale e si arresta il calcolo a $V_2(\cdot)$ quando si vuole risolvere il problema per un budget specifico B. La **politica ottima** $x_t^* = \mu_t^*(s_t)$ non è memorizzata in una tabella esplicita, ma è **implicita** nella sequenza delle funzioni valore.

Implementazione MATLAB: idea generale La funzione `findPolicy` serve a costruire, stadio per stadio, un'**approssimazione** delle funzioni valore tramite **spline cubiche**, usando una griglia uniforme del budget. Riceve il budget totale, una lista di funzioni di profitto (una per attività) e il numero di punti

della griglia. Restituisce una lista di spline che rappresentano le funzioni valore approssimate; la prima funzione non viene usata direttamente quando l'interesse è risolvere il problema per un budget iniziale specifico.

La funzione `applyPolicy` applica la politica ottima **in avanti nel tempo** utilizzando le funzioni valore approssimate: a ogni stadio sceglie l'allocazione che massimizza la somma tra contributo immediato e valore futuro, aggiorna il budget residuo e costruisce il vettore delle allocazioni ottime insieme al valore complessivo ottenuto.

8.3 Controllo stocastico delle scorte

Impostazione del problema Si considera una variazione **stocastica del problema di lot-sizing**, assumendo una **domanda aleatoria discreta**. In questo caso è naturale adottare una rappresentazione **tabellare** della value function. Occorre specificare la **dinamica dello stato** in caso di stockout. Qui si assumono **vendite perse**

$$I_{t+1} = \max\{0,\ I_t + x_t - d_{t+1}\}$$

dove $\big(d_t\big)_{t=1,\dots,T}$ è una sequenza di variabili i.i.d., e x_t è la quantità ordinata al tempo t e consegnata immediatamente.

Al tempo t si osserva l'inventario a mano I_t. Poi si decide l'ordine x_t, che viene ricevuto immediatamente e porta la disponibilità a $I_t + x_t$. Durante l'intervallo $t+1$ si osserva la domanda aleatoria d_{t+1} e si aggiorna l'inventario secondo la dinamica sopra.

Spazio degli stati e azioni ammissibili Per tabulare la value function occorre fissare un limite superiore allo stato. Si assume un *vincolo* $I_t \le I_{\max}$, che implica: $\mathcal{X}(I_t) = \{0, 1, \dots, I_{\max} - I_t\}$. Il **costo immediato** include un *costo lineare* d'ordine $c\,x_t$ e una *penalità quadratica* sull'inventario contabile dopo la domanda del periodo successivo: $\beta\,(I_t + x_t - d_{t+1})^2$. L'**inventario fisico** non può essere negativo ed è determinato dalla dinamica con $\max\{0, \cdot\}$; un **inventario contabile** negativo rappresenta domanda non soddisfatta. Per semplicità, il **costo terminale** è nullo: $F_T(I_T) = 0$.

Si può sostenere che la **penalità complessiva** non dovrebbe essere simmetrica e che si potrebbe definire una funzione lineare a tratti con pendenze diverse. Tuttavia, questo aspetto non è realmente essenziale ai fini illustrativi. Ciò che risulta più rilevante è il fatto che il costo immediato dipende dalla realizzazione del fattore di rischio durante il periodo di tempo $t+1$, successivo alla decisione x_t.

Questo implica che, *nella ricorsione di programmazione dinamica, non compare un termine di costo immediato deterministico della forma* $f_t(s_t, x_t)$*, bensì un termine stocastico della forma* $h_t(s_t, x_t, \xi_{t+1})$.

Ricorsione di programmazione dinamica La **ricorsione di programmazione dinamica** risultante è

$$V_t(I_t) = \min_{x_t \in \mathcal{X}(I_t)} \mathbb{E}_{d_{t+1}}\Big[c\,x_t + \beta\,(I_t + x_t - d_{t+1})^2 + V_{t+1}(\max\{0,\ I_t + x_t - d_{t+1}\})\Big],$$

per $t = 0, 1, \dots, T-1$, con $I_t \in \{0, 1, 2, \dots, I_{\max}\}$.

Modellazione dell'incertezza Poiché i **fattori di rischio** costituiscono semplicemente una sequenza di variabili aleatorie discrete indipendenti e identicamente distribuite, per modellare l'incertezza è sufficiente specificare una *funzione di massa di probabilità*, ossia un vettore di probabilità π_k associato a ciascun possibile valore della domanda $k = 0, 1, 2, \dots, d_{\max}$.

È inoltre necessario specificare lo stato iniziale I_0 e l'orizzonte temporale T considerato per la pianificazione.

Lot-sizing stocastico: idea delle procedure MATLAB La funzione `MakePolicy` serve ad apprendere le funzioni valore **a ritroso** (**backward**) e la **politica ottima** in forma tabellare, dato un limite massimo di inventario, una distribuzione discreta della domanda e i parametri economici. L'output è una tabella dei valori (value function per ogni stato e istante) e una tabella delle azioni (decisione ottima

d'ordine per ogni stato e istante).

La funzione `SimulatePolicy` serve a **simulare** l'applicazione della politica ottima su molti scenari di domanda generati casualmente, per ottenere una stima statistica del costo totale e confrontarla con il valore previsto dalla value function.

8.4 Sfruttare la struttura: cammini minimi per il lot-sizing deterministico

Motivazione Nel problema giocattolo di lot-sizing è semplice memorizzare funzioni valore e politica in una tabella, ma ciò è **inefficiente** quando i valori possibili di domanda sono molti, e diventa impossibile con spazio degli stati **continuo**. Talvolta è possibile semplificare drasticamente il problema sfruttando la **struttura**.

Si consideri una versione deterministica con soli **costi fissi d'ordine** ϕ e **costi di giacenza** h, con inventario iniziale e finale pari a zero, e domanda non nulla nel primo periodo. In linea di principio, una **ricorsione DP** è

$$V_t(I_t) = \min_{x_t \geq d_{t+1} - I_t} \{\phi\, \delta(x_t) + h\,(I_t + x_t - d_{t+1}) + V_{t+1}(I_{t+1})\}\,, \quad t = 0, \ldots, T-1$$

con condizione: $V_T(I_T) \equiv 0$. Il vincolo su x_t garantisce che la domanda sia sempre soddisfatta e che lo stato non diventi negativo.

Teorema: proprietà di Wagner–Whitin Per il **uncapacitated lot-sizing deterministico** con **costi fissi** e **costi lineari di inventario**, esiste una soluzione ottima tale che: $I_t\, x_t = 0, \quad t = 0, 1, \ldots, T-1$. Il **messaggio** è che non è mai ottimale ordinare quando l'inventario è positivo: si ordina solo se l'inventario è **nullo**.

Bilancio globale dei flussi e nodo fittizio Si osserva che, considerando l'intera rete, deve valere una *condizione di equilibrio globale* dei flussi, espressa dalla relazione

$$\sum_{t=0}^{T-1} x_t = \sum_{t=1}^{T} d_t.$$

Tale condizione di equilibrio viene rappresentata introducendo un **nodo fittizio** 0, il cui flusso entrante rappresenta la quantità totale ordinata sull'intero orizzonte di pianificazione.

È inoltre necessario garantire che la *domanda sia soddisfatta in ciascun intervallo temporale*. A tal fine si introduce un insieme di nodi corrispondenti agli istanti temporali $t = 1, \ldots, T$, associati all'ultimo istante di ciascun intervallo, il che equivale ad assumere che la domanda possa essere soddisfatta alla fine dell'intervallo temporale.

Il **bilancio di flusso** al nodo t corrisponde all'equazione di transizione dello stato $I_t = I_{t-1} + x_{t-1} - d_t$. Si assuma ora, in contrasto con il teorema precedente, che valgano contemporaneamente le condizioni $I_{t-1} > 0$ e $x_{t-1} > 0$. In tal caso, **è immediato osservare che, reindirizzando il flusso orizzontale I_{t-1} lungo l'arco di ordinazione associato a x_{t-1}, è possibile ridurre il costo complessivo**.

Conseguenza pratica della condizione di Wagner–Whitin La conseguenza pratica della condizione di Wagner–Whitin è che, all'istante temporale t, **è sufficiente considerare un insieme ristretto di possibili decisioni di ordinazione**. In particolare, l'ordine può coprire esattamente il fabbisogno di uno o più periodi futuri consecutivi, oppure non essere effettuato affatto.

Questo implica che la variabile decisionale può assumere solo valori corrispondenti alla somma delle domande di intervalli temporali consecutivi successivi.

$$x_t \in \left\{0,\ d_{t+1},\ (d_{t+1} + d_{t+2}),\ (d_{t+1} + d_{t+2} + d_{t+3}),\ \ldots,\ \sum_{\tau=t+1}^{T} d_\tau \right\}.$$

Riformulazione come problema di cammino minimo Sfruttando questa proprietà strutturale, il problema a singolo item può essere riformulato come un problema di **cammino minimo** su una rete di dimensioni contenute.

Il nodo iniziale 0 rappresenta lo stato iniziale. Per ciascun istante temporale t, esistono archi che collegano tale nodo agli istanti successivi $t+1, \dots, T$. La soluzione del problema consiste nel determinare un cammino che collega il nodo iniziale al nodo terminale minimizzando il costo complessivo.

Significato economico degli archi Gli archi selezionati nel cammino ottimo rappresentano il numero di intervalli temporali coperti dal successivo ordine.

Il costo associato a ciascun arco tiene conto sia del **costo fisso di ordinazione** sia del **costo di giacenza** generato dall'inventario accumulato per soddisfare le domande future.

Grazie al numero limitato di nodi della rete, la risoluzione del problema di lot-sizing tramite questa formulazione come cammino minimo conduce a un algoritmo **molto efficiente**, con complessità **polinomiale**.

8.5 Lot-sizing stocastico: politiche S e (s, S)

Backlog e penalità convessa La **proprietà di Wagner–Whitin** non vale nel caso stocastico, ma in alcuni casi esistono risultati strutturali. Si assume che non vi siano **vendite perse** e che sia consentito backlog, con costo totale che include:

$$q(s) = h \max\{0, s\} + b \max\{0, -s\},$$

dove s può essere positivo (inventario) o negativo (backlog), h è il costo di giacenza e $b > h$ è il costo di backlog. La funzione $q(\cdot)$ è **convessa** e tende a $+\infty$ quando $s \to \pm\infty$. *Trascurando i costi fissi* e includendo un costo variabile lineare unitario c, l'obiettivo è **minimizzare**

$$\mathbb{E}_0 \left[\sum_{t=0}^{T-1} (c\, x_t + q(I_t + x_t - d_{t+1})) \right].$$

La **ricorsione DP** è

$$V_t(I_t) = \min_{x_t \geq 0} \{ c\, x_t + H(I_t + x_t) + \mathbb{E}[V_{t+1}(I_t + x_t - d_{t+1})] \},$$

dove

$$H(y_t) := \mathbb{E}[q(y_t - d_{t+1})] = h\, \mathbb{E}[\max\{0, y_t - d_{t+1}\}] + b\, \mathbb{E}[\max\{0, d_{t+1} - y_t\}].$$

Si introduce y_t come inventario disponibile dopo l'ordine (lead time nullo): $y_t := I_t + x_t$, con condizione terminale: $V_T(I_T) = 0$. Si assume distribuzione della domanda costante nel tempo. La **ricorsione** si riscrive come

$$V_t(I_t) = \min_{y_t \geq I_t} G_t(y_t) - c\, I_t,$$

dove

$$G_t(y_t) = c\, y_t + H(y_t) + \mathbb{E}[V_{t+1}(y_t - d_{t+1})].$$

Convessità e livello obiettivo S_t Si assume (senza prova) che $V_t(\cdot)$ e $G_t(\cdot)$ siano **convesse** per ogni t, e che $G_t(\cdot) \to +\infty$ quando $y \to \pm\infty$. Ne segue che $G_t(\cdot)$ ha un minimizzatore non vincolato finito

$$S_t = \arg\min_{y_t \in \mathbb{R}} G_t(y_t).$$

L'**esempio** illustra il posizionamento relativo dei minimi **non vincolati** e **vincolati** in un problema di lot-sizing stocastico. **Nel caso (a)**, il livello S_t soddisfa il vincolo $S_t \geq I_t$ e coincide con il minimo non vincolato, cioè $S_t = y_t^*$. In questa situazione, il minimizzatore non vincolato appartiene all'insieme ammissibile e, di conseguenza, il minimo vincolato e quello non vincolato coincidono. **Nel caso (b)**, il

livello S_t non soddisfa il vincolo $S_t < I_t$ e risulta $I_t = y_t^*$. In questo caso, il minimo vincolato è localizzato sul **bordo dell'insieme ammissibile**, poiché il minimo non vincolato non è accessibile a causa del vincolo.

La politica ottima è una politica **base-stock**

$$x_t^* = \mu_t^*(I_t) = \begin{cases} S_t - I_t, & \text{se } I_t < S_t, \\ 0, & \text{se } I_t \geq S_t. \end{cases}$$

I valori S_t sono livelli obiettivo: si ordina quanto serve per raggiungere il target ottimo. In orizzonte finito, il problema è determinare la sequenza ottima di livelli S_t.

Politiche (s, S) con costi fissi Se si introducono costi fissi, si perde convessità. Tuttavia una proprietà correlata (K-convessità) porta a una politica ottima

$$\mu_t^*(I_t) = \begin{cases} S_t - I_t, & \text{se } I_t < s_t, \\ 0, & \text{se } I_t \geq s_t, \end{cases}$$

dipendente da due sequenze s_t e S_t, con $s_t \leq S_t$. In *ambiente stazionario* è ottima una politica stazionaria (s, S). Si ordina solo quando l'inventario è sotto **small** s, riportandolo a **big** S; $S - s$ è una quantità minima d'ordine che aiuta a controllare i costi fissi.

Le maledizioni della programmazione dinamica La **programmazione dinamica** è un principio potente e flessibile, ma presenta alcune limitazioni rilevanti.

La prima è la **maledizione della dimensionalità dello stato**. È necessario disporre della funzione valore per ogni elemento dello spazio degli stati. Quando tale spazio è finito e di dimensioni contenute, le funzioni valore possono essere memorizzate in forma tabellare; tuttavia, questo approccio diventa impraticabile in presenza di spazi degli stati molto grandi.

Una seconda limitazione è la **maledizione dell'ottimizzazione**. La programmazione dinamica viene utilizzata per decomporre un problema multistadio intrattabile in una sequenza di sottoproblemi a singolo stadio. Tuttavia, anche questi sottoproblemi possono risultare complessi e difficili da risolvere.

Si incontra inoltre la **maledizione dell'aspettativa**. Quando i fattori di rischio ξ_t sono rappresentati da variabili aleatorie continue, il calcolo dell'aspettativa richiede la valutazione di integrali multidimensionali complessi; di conseguenza, è necessario adottare qualche strategia di discretizzazione.

Infine, si presenta la **maledizione della modellazione**. Il sistema può essere talmente complesso da rendere impossibile la definizione di un modello esplicito delle transizioni di stato. Questo aspetto è ulteriormente aggravato nel contesto della programmazione dinamica, poiché le transizioni dipendono almeno in parte dalle decisioni di controllo.

9 CAP 9 – Modelli per la programmazione dinamica

Recap La **programmazione dinamica** fornisce un principio generale di **modellazione** per problemi **decisionali multistadio**, fondato sulla **decomposizione** del problema complessivo in una sequenza di sottoproblemi a singolo stadio e sull'uso di una **value function**. Tale funzione consente di bilanciare in modo sistematico la **ricompensa immediata** con il **valore atteso delle decisioni future**, attraverso una ricorsione che incorpora **azioni ammissibili**, **fattori di rischio** e **dinamiche di transizione** dello stato.

Nel caso dei **processi decisionali di Markov finiti**, caratterizzati da spazi di stato e di azione discreti, la dinamica è descritta tramite **probabilità di transizione** dipendenti dall'azione e la ricorsione di DP assume una forma esplicita e computazionalmente ben definita. Tuttavia, l'applicazione pratica è spesso limitata dalla **curse of modeling** e dalla rapida crescita dimensionale dello spazio degli stati.

Per superare tali difficoltà, si introduce la formulazione in termini di **Q-factors** e l'impiego di tecniche di **campionamento Monte Carlo**, che consentono di riscrivere la ricorsione di programmazione dinamica ottenendo uno **scambio tra attesa e ottimizzazione**. Questa riformulazione risulta particolarmente

utile nei problemi di grande scala o con dinamica sconosciuta, aprendo la strada a metodi di apprendimento **model-free** e all'utilizzo di **architetture di approssimazione**.

Un'idea strettamente collegata è l'introduzione dello **stato post-decisione**, che consente di separare la componente deterministica della decisione dall'incertezza legata ai fattori di rischio, riformulando la ricorsione di programmazione dinamica in una forma caratterizzata da **ottimizzazione deterministica** e **attesa esterna**. Questa struttura semplifica sia l'analisi teorica sia lo sviluppo di algoritmi numerici.

Questi principi vengono illustrati attraverso esempi rilevanti, quali l'**aumento dello stato** nei problemi di gestione delle scorte in presenza di **lead time** o articoli deperibili, e i modelli di **revenue management**. In quest'ultimo contesto, nelle versioni statiche e dinamiche, con **segmentazione perfetta** o con **scelta del cliente**, emergono naturalmente **livelli di protezione**, politiche dipendenti dal tempo e diverse forme della ricorsione di DP, determinate dalla struttura informativa e dal timing delle decisioni.

9.1 Principio di modellazione per la programmazione dinamica

Idea generale della programmazione dinamica Il principio della **programmazione dinamica** è un concetto **flessibile** per la **decomposizione** di un problema decisionale multistadio in una sequenza di **problemi a singolo stadio**, bilanciando il contributo immediato e i contributi attesi delle decisioni future. Il principio si fonda sulla **value function**, definita dalla ricorsione di DP

$$V_t(s_t) = \text{opt}_{x_t \in X(s_t)} \{ f_t(s_t, x_t) + \gamma \, \mathbb{E}[\, V_{t+1}(s_{t+1}) \mid s_t, x_t] \} \, .$$

L'equazione precedente è solo una possibile forma della ricorsione di DP. Si possono adottare forme più specifiche, ad esempio quando i **fattori di rischio** sono **variabili aleatorie discrete** e l'attesa si riduce a una somma.

Scambio tra attesa e ottimizzazione In alcuni casi si adotta una riformulazione più radicale, in cui si **scambiano** gli operatori di attesa e ottimizzazione. Ciò può dipendere dalla struttura informativa del problema oppure essere il risultato di una manipolazione volta a evitare la soluzione di un sottoproblema stocastico difficile.

Un caso ben noto è il **Q-learning**, una forma di reinforcement learning in cui $V(s)$ è sostituita da **Q-factors** $Q(s, x)$, che rappresentano il valore delle coppie stato–azione. Più in generale, lo scambio tra ottimizzazione e attesa può essere ottenuto introducendo il concetto di **stato post-decisione**.

Trovare una descrizione adeguata dello stato del sistema può richiedere un'attenta modellazione: può essere necessaria una **ridefinizione dello spazio degli stati** per eliminare dipendenze dal percorso e ottenere un modello **markoviano aumentato ed equivalente**.

Sebbene sia naturale pensare a stati e decisioni come scalari o vettori, essi possono consistere in oggetti diversi, ad esempio insiemi.

Le *capacità di modellazione* procedono di pari passo con la conoscenza algoritmica nell'affrontare un problema decisionale tramite DP; tali capacità si affinano solo con l'esperienza su una vasta gamma di problemi.

Processi decisionali di Markov finiti Il termine **processo decisionale di Markov (MDP)** è riservato a problemi con spazi di stato e spazi delle azioni discreti. Il termine **azione** è spesso adottato per riferirsi alle decisioni di controllo. Stati e azioni discreti possono essere enumerati e associati a numeri interi.

Si considerano **MDP finiti**. Anche se gli stati possono corrispondere a vettori in uno spazio multidimensionale, si usa una notazione del tipo $i, j \in S = \{1, \ldots, N\}$, dove N è la dimensione dello spazio degli stati.

Si usa a o a_t per indicare le azioni; $A_t(i)$ o $A(i)$ denota l'insieme delle **azioni ammissibili** nello stato i al tempo t. L'insieme di tutte le azioni possibili è A.

Il sistema sottostante è una catena di Markov finita a tempo discreto. Nei MDP le **probabilità di transizione** dipendono dall'azione selezionata: $\pi_{t+1}(i, a, j)$ è la probabilità di transizione dallo stato i allo stato j durante l'intervallo $t + 1$, dopo aver scelto l'azione a al tempo t.

Esempio: controllo stocastico di inventario Dall'esempio si evince come, con spazio di stato finito e azioni ammissibili dipendenti dallo stato, le **matrici di transizione** dipendano dall'azione e possano essere rappresentate in forma tabellare.

Ricorsioni DP per MDP Nel contesto MDP, la value function al tempo t è un vettore finito-dimensionale con componenti $V_t(i)$ e soddisfa

$$V_t(i) = \text{opt}_{a\in A(i)} \left\{ f_t(i,a) + \gamma \sum_{j\in S} \pi_{t+1}(i,a,j)\, V_{t+1}(j) \right\}, \quad i \in S.$$

Nel caso a **orizzonte infinito scontato**

$$V(i) = \text{opt}_{a\in A(i)} \left\{ f(i,a) + \gamma \sum_{j\in S} \pi(i,a,j)\, V(j) \right\}, \quad i \in S.$$

Se il **contributo immediato è stocastico** e dipende dallo stato successivo, lo si denota $h(i,a,j)$ e si riscrive

$$V(i) = \text{opt}_{a\in A(i)} \sum_{j\in S} \pi(i,a,j)\ \{h(i,a,j) + \gamma V(j)\}, \quad i \in S.$$

In principio, $f(i,a) = \sum_{j\in S} \pi(i,a,j)\, h(i,a,j)$, ma ciò può essere impraticabile quando N è finito ma enorme oppure quando le probabilità di transizione non sono note.

Esempio: arresto ottimo ricorrente Dall'esempio si evince il **trade-off** tra ricompensa immediata e attesa di opportunità migliori, e come la struttura della catena consenta una formulazione DP esplicita sui valori $V(k)$.

9.2 Valutazione delle politiche e Q-factors

Valutazione delle politiche e Q-factors L'esempio precedente potrebbe suggerire che gli **MDP finiti** siano piuttosto semplici da trattare. La **value function** è semplicemente un vettore e, nel caso di un **orizzonte finito**, esiste una relazione esplicita tra le funzioni valore a stadi diversi. Se l'insieme delle **azioni ammissibili** non è troppo grande, il passo di ottimizzazione si risolve banalmente per enumerazione.

Nel caso a **orizzonte infinito**, è necessario determinare i valori degli stati risolvendo un **sistema di equazioni**, che non è lineare (è infatti **a tratti lineare**). Tuttavia, tale sistema può essere risolto tramite metodi iterativi piuttosto semplici. In realtà, il problema degli MDP potrebbe non essere così semplice, a causa della **dimensione dello spazio degli stati**. Inoltre, anche solo determinare l'intero insieme delle **probabilità di transizione** $\pi(i,a,j)$ può risultare molto difficile, se non impossibile, quando si manifesta la **curse of modeling**.

Un primo ingrediente per aggirare queste difficoltà è il **campionamento Monte Carlo**. Un secondo ingrediente consiste nel riscrivere la **ricorsione di DP** in una forma diversa, basata sui **Q-factors**. Questo viene ora fatto nel caso a **orizzonte infinito**. In modo informale, un **Q-factor** $Q(i,a)$ misura il valore di intraprendere l'azione a quando il sistema si trova nello stato i. Più precisamente, tale quantità deve essere definita con riferimento alla **politica** che verrà applicata negli stadi successivi.

Policy evaluation e policy iteration Una **politica stazionaria ammissibile** μ associa a ciascuno stato i un'azione $a = \mu(i) \in A(i)$. La value function associata a μ si ottiene risolvendo un sistema lineare

$$V^\mu(i) = f\big(i,\mu(i)\big) + \gamma \sum_{j\in S} \pi\big(i,\mu(i),j\big)\, V^\mu(j), \quad i \in S.$$

Questa relazione è fondamentale nei metodi basati su **policy iteration**: si valuta una politica candidata e poi si tenta di migliorarla.

Definizione di Q-factor per una politica Il **Q-factor** associato a μ è la mappa $S \times A \to \mathbb{R}$:

$$Q^{\mu}(i,a) := f(i,a) + \gamma \sum_{j \in S} \pi(i,a,j)\, V^{\mu}(j).$$

L'idea è applicare a nello stato i e poi seguire la politica μ. Sostituendo la value function ottima nella relazione precedente si ottengono i **Q-factors ottimi**

$$Q(i,a) = f(i,a) + \gamma \sum_{j \in S} \pi(i,a,j)\, V(j), \qquad i \in S,\ a \in A(i).$$

Vale $V(j) \equiv \mathrm{opt}_{a \in A(j)}\, Q(j,a)$, $j \in S$. La **ricorsione di DP** può essere riscritta come

$$Q(i,a) = f(i,a) + \gamma \sum_{j \in S} \pi(i,a,j) \left[\mathrm{opt}_{\tilde{a} \in A(j)}\, Q(j,\tilde{a})\right], \quad i \in S,\ a \in A(i).$$

Vantaggi e svantaggi Confrontando le equazioni, emergono sia aspetti negativi sia aspetti positivi. Lo **svantaggio principale** è che, al posto della funzione valore sugli stati $V(i)$, si introducono ora funzioni valore **stato–azione** $Q(i,a)$; salvo casi di dimensione molto ridotta, ciò sembra aggravare ulteriormente la **curse of dimensionality**. Il **vantaggio fondamentale** è però lo **scambio tra attesa e ottimizzazione**. Questo aspetto risulta particolarmente rilevante nei problemi continui, nei quali è spesso più semplice risolvere molti problemi di ottimizzazione deterministici di piccole dimensioni piuttosto che un unico grande problema stocastico. Inoltre, i Q-factors possono essere **appresi tramite campionamento statistico**, caratteristica cruciale sia nei problemi di grande scala sia quando la dinamica sottostante è sconosciuta; ciò apre la strada alla **programmazione dinamica model-free**. Infine, quando la dimensionalità rende impossibile determinare esplicitamente tutti i Q-factors per ogni coppia stato–azione, è possibile ricorrere a una **rappresentazione compatta** basata su un'**architettura di approssimazione**, che può spaziare da una regressione lineare relativamente semplice fino alle **reti neurali profonde**.

Diverse forme di DP stocastica Si consideri nuovamente la *ricorsione di DP*

$$V_t(s_t) = \mathrm{opt}_{x_t \in X(s_t)} \left\{ f_t(s_t, x_t) + \gamma\, \mathbb{E}[\, V_{t+1}(s_{t+1}) \mid s_t, x_t] \right\}.$$

Si osserva lo stato s_t all'istante t, si prende una decisione ammissibile $x_t \in X(s_t)$, si osserva un contributo immediato $f_t(s_t, x_t)$, si passa a un nuovo stato s_{t+1} che dipende dai fattori di rischio realizzati ξ_{t+1}, secondo una distribuzione che può dipendere da s_t e x_t.

Per determinare $V_t(\cdot)$ a partire da $V_{t+1}(\cdot)$ si dovrebbe risolvere un problema di **ottimizzazione stocastica** che può includere un'attesa impegnativa; nei MDP i Q-factors consentono di scambiare ottimizzazione e attesa, e talvolta lo scambio è richiesto dalla struttura informativa.

Esempio: lot-sizing con lookahead limitato Dall'esempio si evince che, quando la domanda del periodo successivo è nota al momento della decisione, il contributo immediato diventa deterministico e la ricorsione di programmazione dinamica assume una forma in cui l'ottimizzazione è interna all'attesa. La struttura della ricorsione riflette quindi direttamente l'ordine temporale tra informazione, decisione e dinamica del sistema.

Variabili di stato post-decisione Una *forma alternativa* è

$$V_t(s_t) = \mathbb{E}_t\left[\mathrm{opt}_{x_t \in X(s_t)} \left\{ f_t(x_t, s_t) + \gamma V_{t+1}(s_{t+1}) \right\}\right].$$

Il **problema di ottimizzazione** interno è deterministico e l'attesa esterna può essere stimata tramite campionamento statistico. Poiché si tratta di vantaggi rilevanti, si può tentare di **riformulare la ricorsione standard di programmazione dinamica** in questa forma, anche quando essa non rappresenta la formulazione più naturale del problema. In alcuni casi, ciò può essere ottenuto riscrivendo la **dinamica**

del sistema tramite l'introduzione di **variabili di stato post-decisione**. Nella rappresentazione consueta della dinamica di un sistema, si considerano transizioni dallo stato s_t allo stato s_{t+1}, che dipendono dalla decisione selezionata x_t e dalla realizzazione del **fattore di rischio** ξ_{t+1}. In alcune situazioni, risulta tuttavia conveniente introdurre uno **stato intermedio**, osservato dopo che la decisione x_t è stata presa ma prima che il fattore di rischio ξ_{t+1} si realizzi. Tale stato prende il nome di **stato post-decisione** e può essere indicato con s_t^x.

Esempio Dall'esempio si evince che l'introduzione dello **stato post-decisione** permette di separare la componente deterministica della decisione da quella stocastica legata al fattore di rischio. In questo modo, l'incertezza viene assorbita in un'**attesa priva di decisioni**, mentre l'ottimizzazione diventa **deterministica**. La ricorsione di programmazione dinamica assume quindi una forma con **scambio tra attesa e ottimizzazione**, che facilita sia l'analisi sia l'approssimazione numerica.

Relazione tra funzioni valore Si definisce il valore del post-decision state $V_t(s_t)$ e $V_t^x(s_t^x)$: $V_t^x(s_t^x) = \mathbb{E}[\, V_{t+1}(s_{t+1}) \mid s_t^x]$. Quindi la **ricorsione standard** può essere riscritta come **ottimizzazione deterministica**

$$V_t(s_t) = \mathrm{opt}_{x_t \in X(s_t)} \left\{ f_t(s_t, x_t) + \gamma V_t^x(s_t^x) \right\}.$$

Scrivendo la relazione un passo indietro e sostituendo l'espressione precedente, si ottiene una ricorsione che presenta nuovamente lo **scambio tra attesa e ottimizzazione**

$$V_{t-1}^x(s_{t-1}^x) = \mathbb{E}\left[V_t(s_t) \;\middle|\; s_{t-1}^x \right] = \mathbb{E}\left[\mathrm{opt}_{x_t \in X(s_t)} \left(f_t(s_t, x_t) + \gamma V_t^x(s_t^x) \right) \;\middle|\; s_{t-1}^x \right].$$

Trascurando lo slittamento temporale privo di conseguenze, si ottiene nuovamente una ricorsione di programmazione dinamica caratterizzata dallo **scambio tra attesa e ottimizzazione**. La ricorsione di DP formulata in termini di **Q-factors** si inserisce naturalmente in questo quadro, poiché la coppia **stato–azione** (i, a) può essere interpretata come uno **stato post-decisione**, prima dell'osservazione della transizione casuale verso il nuovo stato.

Aumento dello stato nella gestione delle scorte Nel contesto della gestione delle scorte, l'inventario **on-hand** appare come la variabile di stato naturale e come la base più immediata per qualsiasi decisione di riordino. Al contrario, le decisioni di ordinazione non dovrebbero essere basate sull'inventario on-hand, bensì sull'**inventario disponibile**, che tiene conto sia dell'inventario **on-order** (articoli ordinati al fornitore ma non ancora consegnati) sia dell'eventuale **backlog**.

L'equazione di stato standard $I_{t+1} = I_t + x_t - d_{t+1}$, assume che quanto ordinato all'istante di tempo t sia immediatamente disponibile per soddisfare la domanda durante l'intervallo di tempo successivo $t+1$.

Se il **lead time** di consegna è un numero intero $LT \geq 1$ di intervalli di tempo, è necessario introdurre variabili di stato che tengano traccia di ciò che è stato ordinato in passato e si trova ancora nella pipeline di trasporto.

Si denotino tali variabili di stato con $z_{t,\tau}$, che rappresentano la quantità che verrà consegnata τ intervalli di tempo dopo l'istante corrente t, dove $\tau = 0, 1, 2, \ldots, LT - 1$. In particolare, $z_{t,0}$ rappresenta ciò che è immediatamente disponibile all'istante di tempo t ed è coinvolto nell'equazione di transizione dell'inventario on-hand: $I_{t+1} = I_t + z_{t,0} - d_{t+1}$, se si trascura l'incertezza sulla domanda.

Ciò che viene ordinato all'istante di tempo t, rappresentato dalla variabile decisionale x_t, sarà disponibile dopo LT intervalli di tempo. Pertanto, all'istante di tempo successivo $t+1$, la quantità x_t si troverà a $LT - 1$ intervalli di tempo dalla consegna. Si può quindi collegare la decisione x_t alla variabile di stato aggiuntiva corrispondente a $\tau = LT - 1$ come segue: $z_{t+1,LT-1} = x_t$. L'equazione di transizione generale per le variabili di stato aggiuntive $z_{t,\tau}$, per $\tau < LT - 1$, si riduce a un semplice **shift temporale**

$$z_{t+1,\tau} = z_{t,\tau+1}, \qquad \tau = 0, 1, \ldots, LT - 2.$$

In modo analogo, nel caso di articoli **deperibili** è necessario descrivere l'inventario a ciascuna età, introducendo un array di variabili di stato $I_{t,\tau}$ che rappresentano la quantità di inventario on-hand al

tempo t con un'età pari a τ periodi di tempo. Le transizioni di stato non sono immediate da scrivere in forma esplicita e dipendono dal tipo di politica di issuing adottata, **FIFO** (*first-in-first-out*) oppure **LIFO** (*last-in-first-out*).

Revenue management Il **revenue management** consiste in una serie di modelli e tecniche finalizzate a massimizzare il **ricavo** ottenuto dalla vendita di **risorse deperibili**. L'*idea* è stata introdotta, con il nome di *yield management*, nell'industria del trasporto aereo.

Si distinguono due approcci fondamentali al revenue management: l'approccio **quantity-based** e l'approccio **price-based**. Nel primo caso, la disponibilità della risorsa viene limitata secondo una certa politica al fine di massimizzare il ricavo; nel secondo caso, i prezzi vengono aggiustati dinamicamente e si parla di **dynamic pricing**.

Si considerano modelli di programmazione dinamica semplici per il revenue management quantity-based, assumendo che il costo marginale di ciascuna unità di risorsa sia nullo o trascurabile, così che la massimizzazione del profitto si riduca alla massimizzazione del ricavo.

Si considerino C unità di una singola risorsa, ad esempio posti su un aeromobile, che devono essere allocate a n classi tariffarie, indicizzate da $j = 1, 2, \ldots, n$. Le unità della classe j sono vendute al prezzo p_j, con $p_1 > p_2 > \cdots > p_n$, così che la classe 1 corrisponde alla classe dalla quale si ottiene il ricavo più elevato.

Si assume che i posti allocati alle diverse classi siano fisicamente **identici**, ma vengano differenziati mediante **servizi accessori** (*ancillaries*), come diritti di cancellazione, restrizioni sul weekend o servizi a bordo, così da offrire pacchetti differenti.

Il prezzo di ciascuna classe è fissato, mentre il controllo avviene sulla **disponibilità di inventario**, limitando il numero di posti offerti a ciascuna classe. La domanda D_j di ciascuna classe è aleatoria e può realizzarsi secondo schemi differenti, dando luogo a formulazioni di problema diverse.

Due sono le caratteristiche essenziali considerate. La prima riguarda il **comportamento dei clienti**: in un mercato perfettamente segmentato, ciascun passeggero è disposto ad acquistare una sola classe e non è necessario modellare le preferenze; al contrario, per tener conto delle preferenze è necessario introdurre un **modello di scelta**. La seconda riguarda il **timing della domanda**: sarebbe auspicabile che la domanda si realizzasse in modo sequenziale, dalla classe 1 fino alla classe n, mentre il caso peggiore si verifica quando i clienti a basso budget arrivano per primi.

Un modello in cui esistono intervalli di tempo disgiunti, durante i quali una sola classe è in domanda, è detto **statico**; il termine **dinamico** è invece riservato a modelli in cui le richieste dei clienti per classi diverse sono intercalate nel tempo. La politica decisionale può essere espressa in termini di **livelli di protezione**, ossia il numero massimo di posti disponibili per ciascuna classe, oppure in termini di **bid-prices**, ovvero il prezzo minimo al quale si è disposti a vendere un posto.

9.3 Modello statico con segmentazione perfetta della domanda

Ipotesi di segmentazione e struttura della domanda Nel modello statico con **segmentazione perfetta della domanda**, le richieste associate alle diverse classi tariffarie sono assunte **indipendenti** e si realizzano in modo **sequenziale**, a partire dalla classe a prezzo più basso. Nessun cliente è disposto a passare a una classe differente, il che implica l'assenza di sostituzione tra prodotti e rende superflua la modellazione delle preferenze individuali.

Stadi decisionali e variabile di stato In questo contesto si associano gli **stadi decisionali** alle classi tariffarie, indicizzate in ordine decrescente $j = n, n-1, \ldots, 1$. La variabile di stato naturale è s_j, che rappresenta la **capacità residua** all'inizio dello stadio j; lo stato iniziale è quindi $s_n = C$, pari al numero totale di posti disponibili.

Funzione valore e condizioni al contorno L'obiettivo complessivo è determinare il **ricavo atteso massimo** ottenibile allocando i C posti alle n classi, valore indicato con $V_n(C)$, mentre la condizione al

contorno è data da $V_0(s_0) = 0$ per ogni $s_0 = 0, 1, \dots, C$. Non si considerano gruppi o famiglie e, data l'indipendenza, non emerge alcun effetto di apprendimento della domanda lungo il processo.

Sequenza degli eventi e formulazione della ricorsione Un risultato apparentemente sorprendente è che la ricorsione di programmazione dinamica può essere costruita assumendo, per ciascuna classe j, una sequenza degli eventi in cui si osserva dapprima la domanda D_j, successivamente si decide quante richieste accettare, rappresentate dalla variabile intera x_j, e infine si incassa il ricavo $p_j x_j$, passando allo stadio successivo con stato aggiornato $s_{j-1} = s_j - x_j$.

Indipendenza della decisione dalla distribuzione della domanda Sebbene ciò sembri in contrasto con il fatto che le decisioni dovrebbero essere prese prima di osservare la domanda, si dimostra che la decisione ottima non dipende dalla distribuzione di probabilità di D_j.

Interpretazione operativa della decisione L'intuizione è che non sia necessario fissare a priori il valore di x_j. Le richieste possono essere osservate una alla volta e, per ciascuna di esse, si decide se accettarla o rifiutarla; nel momento in cui una richiesta viene rifiutata, la classe viene dichiarata chiusa.

In questo modo, la ricorsione di programmazione dinamica assume la **forma scambiata**

$$V_j(s_j) = \mathbb{E}\left[\max_{0 \le x_j \le \min\{s_j, D_j\}} \big(p_j x_j + V_{j-1}(s_j - x_j)\big)\right].$$

Shift di indice e semplificazione della notazione Per semplificare la notazione, si introduce uno **shift inconsequenziale dell'indice** e si omettono i pedici di stadio nelle variabili di stato e di decisione, ottenendo

$$V_{j+1}(s) = \mathbb{E}\left[\max_{0 \le x \le \min\{s, D_{j+1}\}} \big(p_{j+1} x + V_j(s - x)\big)\right].$$

La decisione x allo stadio $j+1$ è naturalmente vincolata sia dalla capacità residua s sia dalla domanda realizzata D_{j+1}.

Valore marginale atteso della capacità Per analizzare la struttura della politica ottima si introduce il **valore marginale atteso della capacità** $\Delta V_j(s) := V_j(s) - V_j(s-1)$, che misura il **costo opportunità** di un posto aereo dato un livello di capacità residua s.

Utilizzando una **somma telescopica**, la ricorsione può essere riscritta come

$$V_{j+1}(s) = V_j(s) + \mathbb{E}\left[\max_{0 \le x \le \min\{s, D_{j+1}\}} \sum_{z=1}^{x} \big(p_{j+1} - \Delta V_j(s + 1 - z)\big)\right].$$

La riscrittura precedente si basa su un **trucco standard del mestiere**, ossia su una **somma telescopica**

$$\begin{aligned} V_j(s - x) = V_j(s) - \big[V_j(s) - V_j(s - x)\big] &= V_j(s) - \big[V_j(s) \pm V_j(s-1) \pm \cdots \pm V_j(s + 1 - x) - V_j(s - x)\big] \\ &= V_j(s) - \sum_{z=1}^{x} \big[V_j(s + 1 - z) - V_j(s - z)\big] = V_j(s) - \sum_{z=1}^{x} \Delta V_j(s + 1 - z). \end{aligned}$$

Proprietà di monotonicità Si può dimostrare che valgono $\Delta V_j(s+1) \le \Delta V_j(s)$ e $\Delta V_{j+1}(s) \ge \Delta V_j(s)$, da cui segue che il valore della capacità decresce con la capacità disponibile e cresce con il numero di stadi rimanenti.

Di conseguenza, i termini $p_{j+1} - \Delta V_j(s + 1 - z)$ sono decrescenti in z, e la soluzione ottima si ottiene aumentando x finché compare il primo termine negativo oppure si raggiunge $\min\{s, D_{j+1}\}$. Questo porta alla definizione di un **livello di protezione annidato ottimo**

$$y_j^* := \max\{y : \; p_{j+1} < \Delta V_j(y)\}, \quad j = 1, \dots, n-1.$$

Decisione ottima e interpretazione Il livello y_j^* vincola il numero massimo di biglietti vendibili per la classe $j+1$, con decisione ottima $x_{j+1}^*(s_{j+1}, D_{j+1}) = \min\{(s_{j+1} - y_j^*)^+,\, D_{j+1}\}$.

Le richieste possono essere soddisfatte progressivamente senza conoscere D_{j+1} in anticipo, il che giustifica lo **scambio tra attesa e ottimizzazione** nella ricorsione di programmazione dinamica.

Modello dinamico con segmentazione perfetta della domanda Si rilassa l'ipotesi secondo cui la domanda delle diverse classi si realizzi in modo sequenziale su intervalli di tempo disgiunti; rimane tuttavia valida l'assunzione di **segmentazione rigida del mercato**. Il tempo entra esplicitamente nella ricorsione di programmazione dinamica, ma si ottiene un modello semplice assumendo che gli intervalli temporali siano sufficientemente piccoli da consentire l'osservazione di **al più un arrivo per intervallo**.

Si denoti con $\lambda_j(t)$ la probabilità di un arrivo della classe j nell'intervallo di tempo t, per $t = 1, \dots, T$. Poiché il mercato è perfettamente segmentato, tali probabilità si riferiscono a eventi indipendenti e devono soddisfare il vincolo di consistenza $\sum_{j=1}^{n} \lambda_j(t) \leq 1$ per ogni t.

La ricorsione DP mira a determinare le funzioni valore $V_t(s)$, dove s rappresenta la **capacità residua**, soggette alle condizioni al contorno $V_t(0) = 0$ per $t = 1, \dots, T$ e $V_{T+1}(s) = 0$ per $s = 0, 1, \dots, C$, dove $T+1$ indica la fine dell'orizzonte temporale.

Si introduca la variabile casuale $R(t)$, che rappresenta il **ricavo disponibile** al tempo t, pari a p_j quando si verifica un arrivo della classe j e pari a zero in caso contrario. In presenza di un arrivo, si deve decidere se accettare o meno la richiesta tramite una variabile decisionale binaria $x \in \{0, 1\}$.

Come nel modello statico, non è necessario prendere decisioni in anticipo, poiché è sufficiente reagire alla richiesta osservata; di conseguenza, la ricorsione di programmazione dinamica assume nuovamente la **forma scambiata**

$$V_t(s) = \mathbb{E}\left[\max_{x \in \{0,1\}} \big(R(t)x + V_{t+1}(s - x)\big)\right].$$

Utilizzando ancora i **valori marginali attesi della capacità**, è possibile definire una politica ottima in termini di **livelli di protezione**, che in questo contesto risultano però **dipendenti dal tempo**.

Modello dinamico con scelta del cliente Nel caso in cui si rilassi l'ipotesi di **mercato perfettamente segmentato**, è necessario introdurre un esplicito **modello di scelta del cliente**. Un approccio consiste nel suddividere il mercato in segmenti e stimare, per ciascun segmento, sia la frazione di passeggeri sia la probabilità che una determinata classe venga acquistata quando essa è offerta.

La **decisione di controllo** all'istante di tempo t non è più una quantità, ma un insieme: il sottoinsieme $S_t \subseteq \mathcal{N} = \{1, \dots, n\}$ delle classi che vengono offerte. A ciascuna classe j è associato un ricavo p_j; si introduce inoltre $p_0 = 0$ per rappresentare il caso di **non-acquisto**, ossia quando un passeggero è disposto a volare ma non sceglie nessuna delle classi offerte.

Formalmente, una potenziale richiesta arriva al tempo t con probabilità λ e, dato l'insieme S_t, il cliente sceglie una classe $j \in S_t \cup \{0\}$ con probabilità $P_j(S_t)$, con vincoli $P_j(S) \geq 0$ e $\sum_{j \in S} P_j(S) + P_0(S) = 1$. Ne segue che le probabilità effettive di acquisto al tempo t sono $\lambda P_j(S_t)$ per $j = 1, \dots, n$ e $(1 - \lambda) + \lambda P_0(S_t)$ per il non-acquisto.

In questo contesto, la **ricorsione di programmazione dinamica** assume la forma

$$V_t(s) = \max_{S_t \subseteq \mathcal{N}} \left\{ \sum_{j \in S_t} \lambda P_j(S_t)\big(p_j + V_{t+1}(s-1)\big) + \big(\lambda P_0(S_t) + 1 - \lambda\big) V_{t+1}(s) \right\},$$

poiché l'offerta S_t deve essere fissata *prima* della decisione del passeggero. Di conseguenza, a differenza dei modelli con segmentazione perfetta, l'ottimizzazione avviene nella forma usuale, come **massimizzazione di un valore atteso**.

10 CAP 10 – Programmazione dinamica numerica per stati discreti

Recap La **programmazione dinamica numerica per stati discreti** riguarda problemi decisionali in cui lo stato evolve a tempo discreto $t = 0, 1, 2, \ldots$ su uno spazio di stati finito o numerabile e la dinamica è descritta da **probabilità di transizione** che rispettano la proprietà di Markov. Nel caso **omogeneo** le probabilità non dipendono dal tempo, ipotesi naturale nei problemi a **orizzonte infinito**; nei **Markov Decision Processes** (MDP) la dinamica è **parzialmente controllata** tramite la scelta di un'azione $a \in \mathcal{A}(i)$ e, per ogni azione, la transizione è descritta da $\pi(i, a, j)$, che consente di rappresentare il modello anche tramite insiemi di matrici di transizione indicizzate dall'azione.

La valutazione delle prestazioni di una politica richiede di distinguere le proprietà della catena indotta: la presenza di **stati assorbenti** genera stati transienti e può rendere la dinamica dipendente dalla condizione iniziale; se la catena non è **irriducibile** si possono ottenere limiti diversi a seconda dello stato di partenza, mentre l'assenza di **aperiodicità** impedisce di definire una distribuzione stazionaria. Per evitare comportamenti degeneri e garantire che gli algoritmi "vedano" gli stati con sufficiente frequenza, si assume che il processo sia **ben comportato** per qualunque politica, tipicamente con struttura **unichain** e senza periodicità, così da rendere sensata l'idea di prestazione di lungo periodo e l'uso di metodi iterativi.

Nei problemi a **orizzonte temporale finito** la soluzione è espressa da una ricorsione di Bellman in cui la funzione di valore $V_t(i)$ dipende esplicitamente da t e bilancia contributo immediato e valore futuro atteso, $V_t(i) = \mathrm{opt}_{a \in \mathcal{A}(i)} \{ f(i, a) + \gamma \sum_j \pi(i, a, j) V_{t+1}(j) \}$, con condizione terminale al bordo che determina il comportamento all'ultimo istante decisionale. Nell'esempio di **arresto ottimo** su catena lineare, l'azione *wait* induce un *random walk* mentre l'azione di stop produce una ricompensa R_i e porta a uno stato assorbente; la struttura locale delle transizioni implica una **matrice di transizione tridiagonale** e consente una forma vettoriale compatta del confronto tra ricompensa e valore di continuazione, evidenziando che la politica ottima può essere **non stazionaria** e dipendere sia dallo stato sia dal tempo residuo.

Nei problemi a **orizzonte infinito** la funzione di valore è invece definita come soluzione di un'equazione di punto fisso del tipo $V(i) = \mathrm{opt}_{a \in \mathcal{A}(i)} \{ f(i, a) + \gamma \sum_j \pi(i, a, j) V(j) \}$, o in forma equivalente tramite contributi immediati $h(i, a, j)$ dentro la somma. Per MDP finiti con **sconto stretto** $\gamma < 1$ e contributi immediati limitati, l'operatore di Bellman $\mathcal{T}$ e l'operatore associato a una politica fissata $\mathcal{T}_\mu$ risultano **contrazioni**, quindi ammettono un unico punto fisso: $\mathbf{V} = \mathcal{T}\mathbf{V}$ caratterizza il valore ottimo e $\mathbf{V}_\mu = \mathcal{T}_\mu \mathbf{V}_\mu$ caratterizza il valore della politica.

Su questa base si introducono due schemi numerici: la **value iteration**, che applica ripetutamente $\mathcal{T}$ ottenendo aggiornamenti poco costosi e una convergenza al limite verso $\mathbf{V}$, e la **policy iteration**, che alterna *policy evaluation* e *policy improvement*. In particolare, fissata μ, la valutazione della politica equivale a risolvere un sistema lineare $(\mathbf{I} - \gamma \Pi_\mu)\mathbf{V}_\mu = f_\mu$, mentre il miglioramento sceglie per ogni stato un'azione in $\tilde{\mu}(i) \in \arg\mathrm{opt}_{a \in \mathcal{A}(i)} \{ f(i, a) + \gamma \sum_j \pi(i, a, j) V_\mu(j) \}$, garantendo che la politica non peggiori e portando a convergenza in un numero finito di passi perché le politiche stazionarie deterministiche sono in numero finito.

Il confronto tra i metodi mette in luce un continuum di approcci intermedi: quando la valutazione esatta tramite eliminazione di Gauss è impraticabile su grandi sistemi o su matrici sparse, si può valutare μ iterando il punto fisso di $\mathcal{T}_\mu$ e interrompere dopo un numero finito di passi ottenendo una stima $\hat{V}_\mu$, applicando poi il miglioramento; questo schema è alla base della **optimistic policy iteration** e, più in generale, della **generalized policy iteration**. La distinzione diventa cruciale nel **reinforcement learning**: se il modello non è noto, si ricorre a metodi *model-free* che sostituiscono V con i **Q-factors** $Q(s, a)$, con approcci *off-policy* come il **Q-learning** e approcci *on-policy* come **SARSA**, in cui non è realistico stimare esattamente il valore di una politica e i passi di miglioramento devono essere eseguiti comunque in modo anticipato.

Catene di Markov a tempo discreto Una **catena di Markov a tempo discreto** è un *processo stocastico* caratterizzato da una variabile di stato s_t, con $t = 0, 1, 2, \ldots$, che assume valori in un **insieme discreto**. Quando lo spazio degli stati è numerabile, è possibile associare gli stati a numeri interi e

rappresentare il processo mediante un *grafo orientato*, in cui le transizioni sono descritte da archi diretti etichettati con le **probabilità di transizione**. Le probabilità di transizione sono *probabilità condizionate* e dipendono esclusivamente dallo stato corrente, secondo la proprietà di Markov; in particolare, nel caso **omogeneo** (invariante nel tempo), esse non dipendono dall'istante temporale, ipotesi tipica nei problemi a *orizzonte infinito*.

La dinamica della catena può essere descritta raccogliendo le probabilità di transizione in una **matrice di transizione a un passo**, in cui ogni riga rappresenta uno stato di partenza e ogni colonna uno stato di arrivo; poiché dopo ogni transizione il processo deve necessariamente trovarsi in uno degli stati dello spazio degli stati, la somma degli elementi di ciascuna riga è pari a uno. Nei *Markov Decision Processes* (MDP), le transizioni sono **parzialmente controllate** dalla scelta di un'azione: a ogni stato i è associato un insieme finito di azioni ammissibili, e per ciascuna azione è definita una distribuzione di probabilità di transizione sugli stati successivi.

Per valutare le prestazioni di una **politica di controllo stazionaria**, si introducono le *probabilità stazionarie di lungo periodo* di trovarsi nei diversi stati, raccolte in un vettore; tali probabilità, tuttavia, possono non esistere se la catena non è sufficientemente *ben comportata*. Dall'esempio si evince che, se la catena non è **irriducibile**, le probabilità di lungo periodo possono dipendere dallo stato iniziale; per questo motivo si assume una catena **unichain**, connessa, in cui ogni stato è raggiungibile da ogni altro in tempo finito con probabilità positiva ed è visitato infinitamente spesso nel lungo periodo.

Nel caso di stati **assorbenti**, una volta raggiunti essi vengono visitati per sempre, mentre gli altri stati risultano *transienti*; al contrario, in una catena priva di stati assorbenti tutti gli stati possono essere *ricorrenti*. In generale, la presenza simultanea di stati ricorrenti e transienti non impedisce necessariamente l'esistenza di una distribuzione stazionaria, ma può creare difficoltà per il funzionamento di alcuni algoritmi di apprendimento, che richiedono che ogni stato venga visitato con sufficiente frequenza. Infine, in presenza di **periodicità** negli stati non è possibile definire una distribuzione stazionaria; per questo motivo, le catene **aperiodiche** sono una condizione fondamentale affinché valgano diversi risultati teorici rilevanti. Nel seguito si assume che il processo di Markov considerato sia ben comportato per qualunque scelta della politica di controllo.

Processi decisionali di Markov a orizzonte temporale finito Una **catena di decisione di Markov a orizzonte finito** è caratterizzata da una ricorsione di **programmazione dinamica** in cui, a ogni istante t, la funzione di valore $V_t(i)$ bilancia il *contributo immediato* associato all'azione scelta nello stato i e il *valore atteso futuro*, calcolato tramite le probabilità di transizione e la funzione di valore allo stadio successivo, secondo la relazione

$$V_t(i) = \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_j \pi(i,a,j)\, V_{t+1}(j) \right\};$$

la funzione di valore dipende quindi esplicitamente dal tempo e dall'orizzonte finale.

Il problema viene illustrato tramite un esempio di **arresto ottimo**, in cui gli stati sono disposti lungo una catena lineare che rappresenta un *random walk*: dagli stati interni è possibile rimanere nello stesso stato, salire o scendere con probabilità assegnate, mentre dagli stati di bordo sono ammesse solo alcune transizioni. A ogni istante temporale si deve scegliere se *attendere*, lasciando evolvere il sistema in modo stocastico senza costi, oppure *fermare il processo*.

Se si sceglie di attendere, lo stato evolve secondo il meccanismo di random walk e le probabilità di transizione coincidono con quelle associate all'azione *wait*, cioè $\pi_{ij} = \pi(i, \text{wait}, j)$. Se invece si sceglie di fermare il processo, si ottiene una **ricompensa dipendente dallo stato** R_i e il sistema entra in uno stato terminale assorbente, nel quale rimane fino all'istante finale dell'orizzonte. Questo meccanismo può essere interpretato come il problema di *vendere in modo ottimale un asset*, il cui prezzo segue un random walk non influenzato dalle decisioni del decisore, mentre lo stato ha natura puramente informativa.

Si osserva una leggera deviazione dalla notazione standard, poiché la decisione è consentita fino all'istante finale T incluso, che rappresenta l'ultimo istante in cui è possibile fermare il processo; ciò implica che la **politica ottima non è stazionaria**. Le funzioni di valore $V_t(i)$ devono essere determinate per tutti gli istanti e tutti gli stati, imponendo come condizione al bordo $V_T(i) = R_i$, cioè che, al tempo

finale, la scelta ottima consista nel fermare immediatamente il processo.

Le equazioni di programmazione dinamica assumono una forma esplicita che confronta, in ciascuno stato, la ricompensa immediata ottenibile fermando il processo con il valore scontato dell'attesa, che negli stati interni può essere scritto come $\gamma\big(\pi_{i,i-1}V_{t+1}(i-1) + \pi_{i,i}V_{t+1}(i) + \pi_{i,i+1}V_{t+1}(i+1)\big)$. Dall'esempio si evince che la struttura delle transizioni induce una **matrice di transizione tridiagonale**, che consente una rappresentazione compatta del problema.

In forma vettoriale, la ricorsione può essere scritta come $\mathbf{V}_t = \max\{\mathbf{R}, \gamma\Pi\mathbf{V}_{t+1}\}$, dove il massimo è inteso componente per componente, evidenziando la natura deterministica dell'operatore di Bellman nel caso a orizzonte temporale finito.

Implementazione numerica della politica ottima La funzione MATLAB `FindPolicyFiniteRW` implementa la *ricorsione di programmazione dinamica* per un problema di **arresto ottimo** a orizzonte temporale finito, dato una matrice di transizione tridiagonale, un vettore di payoff, un orizzonte temporale e un fattore di sconto.

La funzione calcola iterativamente, procedendo all'indietro nel tempo, la **funzione di valore** e la **politica ottima**, confrontando in ogni stato il valore dell'attesa con il payoff immediato.

In uscita si ottengono una tabella dei valori e una tabella binaria delle decisioni, da cui si evince che la *politica risultante non è stazionaria* e che le decisioni di arresto dipendono sia dallo stato sia dal tempo residuo.

Processi decisionali di Markov a orizzonte temporale infinito Nel caso di un **MDP a orizzonte temporale infinito**, la funzione di valore è definita in modo implicito come soluzione di un'equazione di Bellman del tipo

$$V(i) = \operatorname*{opt}_{a\in\mathcal{A}(i)} \big\{ f(i,a) + \gamma \sum_{j} \pi(i,a,j)\, V(j) \big\}, \quad i \in \mathcal{S},$$

oppure, in forma equivalente,

$$V(i) = \operatorname*{opt}_{a\in\mathcal{A}(i)} \sum_{j\in\mathcal{S}} \pi(i,a,j) \big\{ h(i,a,j) + \gamma V(j) \big\}, \quad i \in \mathcal{S}.$$

Poiché si considerano MDP finiti, lo spazio degli stati può essere identificato con un insieme finito di interi $\mathcal{S} = \{1,\dots,n\}$ e la funzione di valore $V : \mathcal{S} \to \mathbb{R}$ può essere rappresentata come un vettore $\mathbf{V} \in \mathbb{R}^n$, con componenti $V(i)$, $i \in \mathcal{S}$. Le equazioni precedenti non forniscono una soluzione in forma chiusa, ma definiscono un problema di punto fisso.

Per risolvere numericamente tali equazioni ed individuare la politica ottima, si introducono due strategie fondamentali: la **value iteration**, basata su iterazioni computazionalmente poco costose che convergono alla funzione di valore ottima solo al limite, e la **policy iteration**, che richiede iterazioni più onerose ma converge in tempo finito nel caso di MDP finiti, grazie al numero finito di politiche ammissibili.

A tal fine, è utile definire due operatori. Dato un generico vettore di valori $\tilde{\mathbf{V}}$, non necessariamente ottimo, si definisce l'operatore di Bellman $\mathcal{T}$ come

$$[\mathcal{T}\tilde{V}](i) = \operatorname*{opt}_{a\in\mathcal{A}(i)} \sum_{j\in\mathcal{S}} \pi(i,a,j) \big\{ h(i,a,j) + \gamma \tilde{V}(j) \big\}, \quad i \in \mathcal{S},$$

che associa a $\tilde{\mathbf{V}}$ un nuovo vettore di valori sullo spazio degli stati.

Dato invece un generico vettore $\tilde{\mathbf{V}}$ e una politica stazionaria μ, non necessariamente ottima, si definisce l'operatore $\mathcal{T}_\mu$ come

$$[\mathcal{T}_\mu\tilde{V}](i) = \sum_{j\in\mathcal{S}} \pi(i,\mu(i),j) \big\{ h(i,\mu(i),j) + \gamma \tilde{V}(j) \big\}, \quad i \in \mathcal{S}.$$

L'operatore $\mathcal{T}$ svolge un ruolo centrale nella value iteration, mentre $\mathcal{T}_\mu$ è alla base della policy iteration.

Si osserva che il vettore di valore ottimo $\mathbf{V}$ è un **punto fisso** dell'operatore $\mathcal{T}$, ossia soddisfa $\mathbf{V} = \mathcal{T}\mathbf{V}$,

mentre la funzione di valore $\mathbf{V}_\mu$ associata a una politica stazionaria μ è il punto fisso dell'operatore $\mathcal{T}_\mu$, cioè $\mathbf{V}_\mu = \mathcal{T}_\mu \mathbf{V}_\mu$.

Dato una politica stazionaria μ, la quantità $V_\mu(i)$ rappresenta la prestazione attesa scontata ottenuta partendo dallo stato i e applicando tale politica. L'esistenza di una funzione di valore e di una politica stazionaria ottima non è garantita in generale, ma è assicurata nel caso di MDP finiti con **sconto stretto** $\gamma < 1$ e contributi immediati limitati, cioè quando esiste una costante M tale che $|h(i,a,j)| \leq M$.

L'ingrediente chiave delle dimostrazioni è il fatto che sia $\mathcal{T}$ sia $\mathcal{T}_\mu$ sono **operatori di contrazione**, e quindi ammettono un unico punto fisso. Ai fini applicativi, è sufficiente osservare che l'operatore $\mathcal{T}$ fornisce una caratterizzazione della politica stazionaria ottima e che, tramite una singola operazione di ottimizzazione, consente di migliorare una politica stazionaria non ottima, mentre l'operatore $\mathcal{T}_\mu$ permette di calcolarne la funzione di valore associata.

10.1 Value iteration

Metodo di iterazione del valore Secondo l'equazione $\mathbf{V} = \mathcal{T}\mathbf{V}$, nel caso di MDP finiti con sconto stretto $\gamma < 1$ il problema può essere affrontato ricercando un **punto fisso** dell'operatore di Bellman $\mathcal{T}$. A tal fine, si considera un problema a orizzonte finito con funzione di valore terminale assegnata $\mathbf{V}^{(0)}$ e si applica l'operatore $\mathcal{T}$ ottenendo una nuova funzione di valore

$$V^{(1)}(i) = [\mathcal{T}V^{(0)}](i) = \operatorname*{opt}_{a \in \mathcal{A}(i)} \sum_{j \in \mathcal{S}} \pi(i,a,j)\big\{h(i,a,j) + \gamma V^{(0)}(j)\big\}, \quad i \in \mathcal{S},$$

dove il pedice indica il numero di passi rimanenti. La procedura può essere iterata, mappando $\mathbf{V}^{(k)}$ in $\mathbf{V}^{(k+1)}$ secondo

$$V^{(k+1)}(i) = [\mathcal{T}V^{(k)}](i) = \operatorname*{opt}_{a \in \mathcal{A}(i)} \sum_{j \in \mathcal{S}} \pi(i,a,j)\big\{h(i,a,j) + \gamma V^{(k)}(j)\big\}, \quad i \in \mathcal{S},$$

che può essere interpretata come la ricorsione di un MDP a orizzonte finito con funzione di valore terminale $\mathbf{V}^{(0)}$.

L'intuizione suggerisce che, in presenza di sconto stretto, il contributo della funzione di valore terminale $\mathbf{V}^{(0)}$ diventi trascurabile al crescere di k, poiché è moltiplicato da un fattore γ^k che tende a zero. Se la successione $\{\mathbf{V}^{(k)}\}$, interpretando k come contatore delle iterazioni, converge a un limite $\mathbf{V}$, allora tale limite costituisce un punto fisso dell'operatore $\mathcal{T}$.

Più in generale, dato un operatore $H(\cdot) : \mathbb{R}^n \to \mathbb{R}^n$, la ricerca di una soluzione dell'equazione di punto fisso $\mathbf{y} = H(\mathbf{y})$ può essere affrontata mediante lo schema iterativo $\mathbf{y}^{(k+1)} = H(\mathbf{y}^{(k)}), \quad k = 0, 1, 2, \ldots$, a partire da una stima iniziale $\mathbf{y}^{(0)}$; tuttavia, in assenza di ulteriori proprietà, non vi è garanzia di convergenza né di esistenza o unicità del punto fisso. Nel caso in esame, il fatto che l'operatore $\mathcal{T}$ sia un **operatore di contrazione** assicura invece l'esistenza e l'unicità del punto fisso, giustificando teoricamente l'algoritmo di **value iteration**.

Algorithm 2 Value iteration (MDP finito con sconto stretto)

1: Scegliere una funzione di valore iniziale $V^{(0)}$; se non vi sono indicazioni, porre $V^{(0)}(i) = 0$ per ogni $i \in \mathcal{S}$.
2: Scegliere un parametro di tolleranza ϵ.
3: Porre $k = 0$ e `stop` $=$ `false`.
4: **while** `stop` $\neq$ `true` **do**
5: **for all** $i \in \mathcal{S}$ **do**
6: Calcolare

$$V^{(k+1)}(i) = \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j)\, V^{(k)}(j) \right\}.$$

7: **end for**
8: **if** $\left\| V^{(k+1)} - V^{(k)} \right\|_\infty < \epsilon$ **then**
9: `stop` $=$ `true`.
10: **else**
11: $k = k + 1$.
12: **end if**
13: **end while**
14: Porre $\hat{V} = V^{(k+1)}$.
15: Trovare la politica ottima stimata scegliendo un'azione arbitraria se l'insieme delle azioni ottime non è un singleton

$$\hat{\mu}(i) \in \arg \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j)\, \hat{V}(j) \right\}, \quad i \in \mathcal{S}.$$

16: Restituire $\hat{V}$ e $\hat{\mu}$.

Idea dell'esempio numerico Si considera un semplice problema di **arresto ottimo ricorrente** modellato come una catena di Markov con quattro stati, in cui il decisore può scegliere tra le azioni *wait* e *reset*. Avanzando lungo la catena, la probabilità di raggiungere stati migliori diminuisce, mentre i payoff crescono con lo stato, rendendo il trade-off tra attesa e arresto fortemente dipendente dal **fattore di sconto** γ. L'esempio mostra come, al variare delle probabilità di transizione e di γ, la politica ottima passi da una strategia *greedy* (arresto immediato) a politiche più pazienti, evidenziando chiaramente il ruolo del discounting nella value iteration.

Idea delle funzioni MATLAB La funzione `PW311_FindPolicy` implementa la **value iteration** sfruttando la struttura specifica del problema: è sufficiente descrivere le probabilità di permanenza nello stato e confrontare, a ogni iterazione, il valore dell'attesa con quello del reset. L'algoritmo restituisce la funzione di valore ottima, la politica binaria associata e il numero di iterazioni necessarie alla convergenza. La funzione `PW311_Simulate` utilizza invece una **simulazione Monte Carlo** per stimare le prestazioni scontate di una politica fissata, consentendo di confrontare i valori stimati con quelli teorici e di valutarne la robustezza rispetto a possibili errori di modellazione.

10.2 Policy iteration

Idea e formulazione matematica Con la **policy iteration** può accadere che, a differenza della value iteration, non ci si accontenti di individuare rapidamente una politica ottima basata su valori approssimati degli stati, ma si miri a valutare in modo *esatto* il valore di una politica fissata. Si consideri l'operatore T_μ associato a una politica stazionaria μ, definito come

$$[T_\mu \tilde{V}](i) = f\big(i, \mu(i)\big) + \gamma \sum_{j \in \mathcal{S}} \pi\big(i, \mu(i), j\big) \tilde{V}(j), \qquad i \in \mathcal{S}.$$

A differenza dell'operatore di Bellman T, l'operatore T_μ non coinvolge alcuna ottimizzazione ed è interamente determinato dalla politica μ. La funzione di valore V_μ associata alla politica μ è un punto fisso di T_μ e soddisfa, per ogni $i \in \mathcal{S}$, il sistema di equazioni lineari

$$V_\mu(i) = f\big(i, \mu(i)\big) + \gamma \sum_{j \in \mathcal{S}} \pi\big(i, \mu(i), j\big) V_\mu(j).$$

Raccogliendo le probabilità di transizione indotte da μ nella matrice

$$\Pi_\mu := \begin{bmatrix} \pi(1,\mu(1),1) & \pi(1,\mu(1),2) & \cdots & \pi(1,\mu(1),n) \\ \pi(2,\mu(2),1) & \pi(2,\mu(2),2) & \cdots & \pi(2,\mu(2),n) \\ \vdots & \vdots & \ddots & \vdots \\ \pi(n,\mu(n),1) & \pi(n,\mu(n),2) & \cdots & \pi(n,\mu(n),n) \end{bmatrix},$$

e i contributi immediati nel vettore

$$f_\mu := \begin{bmatrix} f(1,\mu(1)) \\ f(2,\mu(2)) \\ \vdots \\ f(n,\mu(n)) \end{bmatrix},$$

il problema di valutazione della politica può essere scritto in forma compatta come $V_\mu = f_\mu + \gamma \Pi_\mu V_\mu$, ossia $(\mathbf{I} - \gamma \Pi_\mu) V_\mu = f_\mu$, dove $\mathbf{I} \in \mathbb{R}^{n \times n}$ è la matrice identità. Formalmente, la soluzione è data da $V_\mu = (\mathbf{I} - \gamma \Pi_\mu)^{-1} f_\mu$.

Sebbene l'inversione matriciale fornisca una soluzione chiusa, per problemi di grandi dimensioni essa risulta computazionalmente onerosa e distrugge la sparsità della matrice, rendendo preferibili metodi iterativi. Una volta calcolato V_μ, è possibile migliorare la politica definendo

$$\tilde{\mu}(i) \in \arg \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) V_\mu(j) \right\}, \qquad i \in \mathcal{S}.$$

Questa operazione, detta **policy improvement**, garantisce che la nuova politica non sia peggiore della precedente, cioè $V_\mu(i) \leq V_{\tilde{\mu}}(i),\ i \in \mathcal{S}$, nel caso di un problema di massimizzazione, con disuguaglianza stretta in almeno uno stato se μ non è ottima. Poiché in un MDP finito il numero di politiche stazionarie deterministiche è finito, una sequenza di valutazioni e miglioramenti conduce necessariamente a una politica ottima, che costituisce il fondamento dell'algoritmo di policy iteration.

Algorithm 3 Policy iteration

1: Definire una politica stazionaria iniziale arbitraria $\mu^{(0)}$.
2: Porre $k = 0$ e `stop = false`.
3: **while** `stop` $\neq$ `true` **do**
4: Valutare la politica $\mu^{(k)}$ risolvendo

$$\left(\mathbf{I} - \gamma \Pi_{\mu^{(k)}}\right) V_{\mu^{(k)}} = f_{\mu^{(k)}}.$$

5: Trovare una nuova politica stazionaria $\mu^{(k+1)}$ tramite *policy improvement*

$$\mu^{(k+1)}(i) \in \arg \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) V_{\mu^{(k)}}(j) \right\}, \qquad i \in \mathcal{S}.$$

6: **if** $\mu^{(k+1)} = \mu^{(k)}$ **then**
7: `stop = true`.
8: **else**
9: $k \leftarrow k + 1$.
10: **end if**
11: **end while**
12: Restituire $V_{\mu^{(k)}}$ e $\mu^{(k)}$.

Idea dell'implementazione MATLAB La funzione `FindPolicyPI` realizza una versione generale e diretta della *policy iteration*, alternando esplicitamente le fasi di *policy evaluation* e *policy improvement*. A partire da una politica iniziale `initialPolicy`, a ogni iterazione viene costruito il sistema lineare $(\mathbf{I} - \gamma \Pi_\mu) V_\mu = f_\mu$ tramite la funzione annidata `makeArrays` e risolto con eliminazione di Gauss per ottenere la funzione di valore della politica corrente. Successivamente, la politica viene migliorata stato per stato scegliendo l'azione che minimizza o massimizza il valore atteso, tramite `findMin` o `findMax`, fino a quando la politica non cambia più. L'output finale consiste nel vettore `stateValues`, che approssima V_μ, e nel vettore `policy`, che rappresenta la politica stazionaria ottima.

Esempio numerico Dall'esempio numerico si evince che la **policy iteration** converge in un numero molto ridotto di passi rispetto alla value iteration. Partendo dalla politica iniziale, che prescrive di attendere ogniqualvolta possibile, la valutazione della politica e il successivo passo di *policy improvement* portano immediatamente alla nuova politica, che risulta già ottima. Questo comportamento conferma che, una volta calcolata esattamente la funzione di valore V_μ, il miglioramento della politica può produrre variazioni significative in un singolo passo, evidenziando l'efficienza concettuale della policy iteration nel caso di MDP finiti con struttura semplice.

Value iteration vs. policy iteration A prima vista, la **value iteration** e la **policy iteration** appaiono molto diverse. La value iteration si basa su un numero potenzialmente elevato di iterazioni computazionalmente poco costose, mentre la policy iteration si fonda su un numero (auspicabilmente ridotto) di iterazioni che possono essere computazionalmente onerose. Inoltre, la convergenza della value iteration è garantita solo al limite, mentre per la policy iteration si ottiene una convergenza in tempo finito. È tuttavia possibile colmare il divario tra i due approcci. Si consideri la valutazione di una politica stazionaria μ mediante iterazione del punto fisso dell'operatore T_μ, secondo

$$V_\mu^{(k+1)}(i) = f\big(i, \mu(i)\big) + \gamma \sum_{j \in \mathcal{S}} \pi\big(i, \mu(i), j\big) V_\mu^{(k)}(j), \qquad i \in \mathcal{S}.$$

In pratica, questa procedura è necessaria quando l'*eliminazione di Gauss* per la risoluzione di grandi sistemi lineari non è applicabile. Idealmente, prima del passo di miglioramento si dovrebbe attendere

la convergenza completa dello schema precedente; tuttavia, se si interrompe prematuramente l'iterazione assumendo in modo ottimistico di aver valutato correttamente la politica corrente, si ottiene una stima $\hat{V}_\mu(i) = V_\mu^{(k+1)}(i)$ e si applica il passo di miglioramento

$$\tilde{\mu}(i) \in \arg \operatorname*{opt}_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) \hat{V}_\mu(j) \right\}, \qquad i \in \mathcal{S}.$$

Questo approccio prende il nome di **optimistic policy iteration** ed è alla base della cosiddetta **generalized policy iteration**. A seconda del grado di ottimismo nella valutazione della politica, l'equazione precedente può essere iterata per un numero elevato o ridotto di passi. Nella value iteration, la politica viene invece aggiornata implicitamente a ogni iterazione tramite

$$V^{(k+1)}(i) = \operatorname*{opt}_{a \in \mathcal{A}(i)} \sum_{j \in \mathcal{S}} \pi(i,a,j) \big\{ h(i,a,j) + \gamma V^{(k)}(j) \big\}, \qquad i \in \mathcal{S},$$

applicando ripetutamente l'operatore T senza mai valutare esplicitamente T_μ. Al contrario, la policy iteration applica molte iterazioni di T_μ prima di passare a una nuova politica. Pertanto, value iteration e policy iteration possono essere viste come gli estremi di un continuo di approcci correlati.

Questa distinzione diventa particolarmente rilevante nel contesto del **reinforcement learning**. Quando le probabilità di transizione (e i contributi immediati) non sono note, si ricorre a una programmazione dinamica *model-free*, in cui la funzione di valore è sostituita dai **Q-factors** $Q(s,a)$ dipendenti sia dallo stato sia dall'azione. In questo contesto, il **Q-learning** rappresenta il corrispettivo della value iteration ed è uno schema di apprendimento *off-policy*, mentre i metodi ispirati alla policy iteration sono tipicamente *on-policy*, come nel caso dell'algoritmo **SARSA**. In reinforcement learning non è in genere possibile apprendere esattamente la funzione di valore di una politica, soprattutto se ciò richiede simulazioni Monte Carlo costose o esperimenti online, rendendo inevitabile l'esecuzione anticipata di passi di miglioramento e dando origine a una vasta famiglia di strategie di apprendimento.

11 CAP 11 – Programmazione dinamica approssimativa e apprendimento per rinforzo per stati discreti

Recap La **programmazione dinamica approssimativa (ADP)** raccoglie metodi numerici che rinunciano alla garanzia di ottimalità per attenuare le **maledizioni della DP** e rendere affrontabili MDP finiti a **orizzonte infinito** anche in assenza di un modello esplicito. Nel contesto **model-free**, le versioni approssimate di value iteration e policy iteration rientrano nel **reinforcement learning (RL)**, in cui un agente apprende interagendo con il sistema osservando stato, contributo immediato e stato successivo. Un punto chiave è che la sola funzione di valore sugli stati non basta per valutare le azioni senza informazioni di transizione; per questo si introducono i **valori stato–azione** tramite **Q-factor**, assumendo (quando possibile) una rappresentazione tabellare oppure ricorrendo a rappresentazioni compatte.

L'apprendimento tramite campionamento porta a due difficoltà strutturali, il compromesso tra **exploration** ed **exploitation** e la **non stazionarietà** del bersaglio, dovuta al fatto che la politica evolve durante l'apprendimento. Per gestire il compromesso si impiegano politiche casuali, come ε-greedy (statico o dinamico) e **Boltzmann exploration** basata su soft-max, mentre la non stazionarietà motiva aggiornamenti incrementali con **smoothing esponenziale** e learning rate costante o decrescente. L'esempio dei **multi-armed bandits** mostra come campioni piccoli e varianza elevata rendano facile selezionare azioni subottimali, giustificando la necessità di exploration controllata.

Nel quadro dei metodi a **temporal differences** si ottiene **SARSA**, schema **on-policy** che apprende i Q-factor di una politica stazionaria tramite correzioni smussate dal learning rate e procedure di **bootstrapping**; l'aggiornamento della politica prima di una valutazione esatta conduce alla **generalized (optimistic) policy iteration**, che può accelerare l'apprendimento ma indebolire la convergenza. In alternativa, il **Q-learning** apprende direttamente i Q-factor **ottimi** con logica **off-policy**, utilizzando un target che combina contributo immediato e valore ottimo stimato dello stato successivo, e aggiornamenti

in stile smoothing esponenziale o temporal difference. La differenza concettuale con SARSA emerge nel fatto che SARSA aggiorna rispetto all'azione scelta dalla politica corrente, mentre Q-learning usa l'azione ottimale (stimata) nello stato successivo.

Le implementazioni MATLAB evidenziano la separazione tra algoritmo e simulatore tramite un oggetto di sistema (`systemObj`/`systemClass`) che fornisce contributo e transizione, il conteggio delle visite stato–azione e la possibilità di integrare una scelta **ε-greedy**. L'esempio numerico mostra che l'esito pratico del Q-learning dipende in modo critico da inizializzazione, learning rate e exploration: senza exploration alcune azioni possono non essere mai provate e i relativi Q-factor non vengono aggiornati, mentre una strategia ε-greedy può recuperare la politica ottima, a costo di maggiore effort computazionale.

ADP e programmazione dinamica model-free La **approximate dynamic programming (ADP)** è un termine ombrello che identifica un'ampia classe di metodi numerici di programmazione dinamica che, a differenza dei metodi standard, **non garantiscono il raggiungimento di una politica ottima**. In cambio della loro natura approssimata, tali metodi mirano ad attenuare le ben note **maledizioni della DP**, rendendo trattabili problemi che sarebbero altrimenti computazionalmente proibitivi. Si considerano qui metodi approssimati per **MDP finiti** nel contesto a **orizzonte infinito**.

Le versioni **model-free** sia della value iteration sia della policy iteration rientrano collettivamente nel quadro del **reinforcement learning (RL)**. In tale impostazione, un **agente** interagisce con un sistema al fine di apprendere una politica di controllo adeguata: non si dispone di un modello del sistema, ma si osservano lo stato corrente e l'esito dell'azione scelta, costituito dal contributo immediato e dallo stato successivo. Una buona politica deve bilanciare obiettivi di breve e di lungo periodo, valutati rispettivamente tramite il contributo immediato e il valore dello stato futuro.

Un punto cruciale del RL model-free è che, anche disponendo di una conoscenza perfetta della funzione di valore degli stati, non è possibile valutare correttamente un'azione in assenza di informazioni sulle transizioni. Per ovviare a questo limite si introducono i **valori stato–azione**, sotto forma di **Q-factor**, che consentono di apprendere una politica decisionale tramite strategie di campionamento. Si assume inizialmente che lo spazio stato–azione non sia eccessivamente grande e che una rappresentazione tabellare dei Q-factor sia praticabile; nella maggior parte dei casi, tuttavia, ciò non è vero e diventa necessario ricorrere a rappresentazioni compatte.

Campionamento e stima in contesti non stazionari Nella teoria della probabilità si è spesso interessati al **valore atteso** di una funzione $h(\cdot)$ di una variabile aleatoria o di un vettore di variabili aleatorie, definito come

$$\theta = \mathbb{E}[h(X)] = \int_{\mathcal{X}} h(x) f_X(x)\, dx,$$

dove $\mathcal{X}$ rappresenta il supporto della distribuzione.

Poiché la valutazione di integrali ad alta dimensionalità risulta in generale complessa, si ricorre a tecniche di **campionamento Monte Carlo**, che permettono di approssimare il valore atteso mediante una media campionaria, ossia

$$\hat{\theta} = \frac{1}{m} \sum_{k=1}^{m} h\big(X^{(k)}\big)\,.$$

Tale stima risulta **non distorta**, in quanto il suo valore atteso coincide con il valore vero di θ.

Nel contesto dei **processi decisionali di Markov**, tuttavia, l'obiettivo non è la stima di una quantità statica, bensì dei **valori di stato** $V(i)$ o dei **Q-factor** $Q(i,a)$. In questa impostazione, l'apprendimento statistico deve confrontarsi con due principali difficoltà: da un lato la necessità di bilanciare **exploration** e **exploitation**, dall'altro la complessità legata alla stima di un **bersaglio non stazionario**, che evolve nel tempo man mano che la politica viene aggiornata.

Il compromesso tra exploration ed exploitation Si supponga di voler apprendere i **Q-factor** $Q(i,a)$ tramite campionamento e di trovarsi attualmente nello stato i. Sorge allora il problema di stabilire quale azione selezionare. Una possibilità consiste nel fare affidamento sulle stime correnti dei Q-factor e scegliere

l'azione che appare più promettente. Questa strategia risulta sensata solo quando il valore associato a ciascuna azione nello stato considerato è stato stimato con sufficiente accuratezza.

All'inizio del processo di apprendimento, tuttavia, tale condizione è difficilmente soddisfatta. Può infatti esistere un'azione che sembra poco conveniente semplicemente perché il suo contributo immediato è stato valutato in modo impreciso; inoltre, tale azione potrebbe condurre a uno stato favorevole, del quale non si ha ancora conoscenza. Queste considerazioni conducono al problema fondamentale del bilanciamento tra **exploration** delle alternative poco conosciute ed **exploitation** delle informazioni attualmente disponibili.

Per chiarire il concetto, si consideri un problema di apprendimento statico, in cui si deve scegliere un'azione da un insieme finito A di alternative. A ciascuna azione $a \in A$ è associata una ricompensa aleatoria $R(a)$. Se fossero noti i valori attesi $\nu(a) = \mathbb{E}[R(a)]$ per ogni azione, in un contesto di neutralità al rischio sarebbe sufficiente ordinare le azioni in base ai rispettivi valori attesi e selezionare quella più promettente.

Si supponga invece di poter ottenere solo stime rumorose $\hat{\nu}(a)$ tramite campionamento. Seguendo una strategia di **pura exploitation**, si sceglierebbe l'azione che massimizza $\hat{\nu}(\cdot)$; tuttavia, una tale scelta può rivelarsi inadeguata, poiché trascura l'incertezza associata alle stime e il potenziale informativo derivante dall'exploration di azioni meno conosciute.

All'estremo opposto rispetto alla pura exploitation si colloca la **pura exploration**, che consiste nella selezione casuale delle azioni secondo probabilità uniformi. È evidente come nessuna di queste due strategie estreme risulti soddisfacente, rendendo necessario individuare un compromesso ragionevole tra exploration ed exploitation.

Una prima possibilità è rappresentata dall'approccio **ε-greedy statico**, in cui si fissa una probabilità ε e si seleziona l'azione più promettente con probabilità $1-\varepsilon$, mentre con probabilità ε si sceglie un'azione casuale. In questo modo si introduce una componente esplorativa controllata, pur privilegiando le azioni che appaiono migliori sulla base delle stime correnti.

Un raffinamento naturale consiste nell'adottare un approccio **ε-greedy dinamico**, in cui il parametro ε viene fatto variare nel tempo, ad esempio ponendo $\varepsilon^{(k)} = \frac{c}{d+k}$ oppure $\varepsilon^{(k)} = d + \frac{c}{k}$ all'iterazione k. In tal modo l'exploration risulta più marcata nelle fasi iniziali dell'apprendimento e viene progressivamente ridotta man mano che le stime diventano più affidabili.

Un'alternativa più sofisticata è fornita dalla **Boltzmann exploration**, che definisce una politica stocastica mediante una funzione soft-max, assegnando a ciascuna azione $a \in A$ una probabilità

$$\varepsilon(a) = \frac{\exp\big(\rho\,\hat{\nu}(a)\big)}{\sum_{a' \in A} \exp\big(\rho\,\hat{\nu}(a')\big)}.$$

Il parametro ρ regola il compromesso tra exploration ed exploitation: per $\rho = 0$ si ottiene una scelta puramente casuale, mentre nel limite $\rho \to +\infty$ la politica converge verso una pura exploitation.

Si osserva quindi che, per gestire il compromesso tra exploration ed exploitation, si ricorre in modo sistematico a **politiche casuali**, nelle quali la selezione delle azioni incorpora deliberatamente una componente stocastica.

Osservazioni sull'esempio dei multi-armed bandits Dall'esempio numerico si evince come, in un problema di **multi-armed bandit**, l'azione ottima in termini di valore atteso possa risultare difficile da individuare quando il numero di osservazioni è limitato. In particolare, sebbene il terzo braccio presenti il valore atteso più elevato, esso è anche caratterizzato da una variabilità significativamente maggiore rispetto agli altri.

A causa dell'elevata varianza, le medie campionarie calcolate su pochi campioni possono discostarsi sensibilmente dal valore atteso, producendo stime che rendono temporaneamente più attrattivi bracci subottimali. Ne consegue che, con una dimensione campionaria ridotta, è altamente probabile selezionare un'azione diversa da quella realmente ottima.

L'esempio mette quindi in evidenza il ruolo cruciale dell'incertezza statistica nella fase di apprendimento e mostra come una strategia basata esclusivamente sulle stime correnti possa condurre a decisioni

errate. Ciò giustifica la necessità di meccanismi che favoriscano un'adeguata exploration, soprattutto nelle fasi iniziali, al fine di ridurre il rischio di convergere prematuramente verso scelte subottimali.

Non stazionarietà e smoothing esponenziale Anche qualora il sistema sottostante sia stazionario, la politica utilizzata per prendere decisioni non lo è, poiché viene aggiornata progressivamente durante il processo di apprendimento. A titolo di confronto, si consideri la media campionaria utilizzata per stimare il valore atteso $\theta = \mathbb{E}[X]$ di una variabile aleatoria scalare X, nella quale tutte le osservazioni raccolte contribuiscono con lo stesso peso.

Indicando con $\hat{\theta}^{(m)}$ la stima ottenuta dopo m osservazioni, si ha

$$\hat{\theta}^{(m)} = \frac{1}{m}\sum_{k=1}^{m} X^{(k)} = \frac{1}{m}\left(X^{(m)} + \sum_{k=1}^{m-1} X^{(k)}\right) = \frac{1}{m}\left(X^{(m)} + (m-1)\hat{\theta}^{(m-1)}\right)$$

$$= \frac{1}{m}X^{(m)} + \frac{m-1}{m}\hat{\theta}^{(m-1)} = \hat{\theta}^{(m-1)} + \frac{1}{m}\left(X^{(m)} - \hat{\theta}^{(m-1)}\right).$$

Quando si osserva un nuovo dato $X^{(m)}$, la stima precedente viene quindi corretta di una quantità proporzionale all'errore di previsione, con un fattore di smoothing pari a $1/m$, che tende ad annullarsi all'aumentare di m.

In un contesto non stazionario può invece risultare opportuno mantenere costante l'entità della correzione, introducendo un coefficiente $\alpha \in (0,1)$ e ponendo

$$\hat{\theta}^{(m)} = \hat{\theta}^{(m-1)} + \alpha\left(X^{(m)} - \hat{\theta}^{(m-1)}\right) = \alpha X^{(m)} + (1-\alpha)\hat{\theta}^{(m-1)}.$$

Questo schema prende il nome di **smoothing esponenziale**.

Sviluppando ricorsivamente l'espressione precedente si ottiene

$$\hat{\theta}^{(m)} = \alpha X^{(m)} + (1-\alpha)\hat{\theta}^{(m-1)} = \alpha X^{(m)} + \alpha(1-\alpha)X^{(m-1)} + (1-\alpha)^2\hat{\theta}^{(m-2)}$$

$$= \sum_{k=0}^{m-1} \alpha(1-\alpha)^k X^{(m-k)} + (1-\alpha)^m \hat{\theta}^{(0)},$$

dove $\hat{\theta}^{(0)}$ è una stima iniziale. A differenza della media campionaria standard, il peso delle osservazioni passate decresce esponenzialmente. Il coefficiente α è noto come **learning rate**, o coefficiente di smoothing, o fattore di dimenticanza.

Valori elevati di α producono aggiornamenti più reattivi ma anche più instabili, mentre valori piccoli introducono maggiore inerzia e filtraggio del rumore. Se la non stazionarietà è dovuta esclusivamente al processo di apprendimento e non a variazioni nella dinamica del sistema, è possibile considerare una sequenza decrescente di learning rate (step sizes) $\alpha^{(k)}$, da applicare al passo k, tale che

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty \quad \text{e} \quad \sum_{k=1}^{\infty} \left[\alpha^{(k)}\right]^2 < \infty.$$

Learning tramite temporal differences e algoritmo SARSA Si è introdotto l'operatore $\mathcal{T}_\mu$, definito come

$$[\mathcal{T}_\mu \tilde{V}](i) = \sum_{j \in S} \pi(i,\mu(i),j)\big(h(i,\mu(i),j) + \gamma \tilde{V}(j)\big),$$

dove μ è una politica stazionaria e $\tilde{V}$ è una funzione di valore. Il valore V_μ associato alla politica μ può essere ottenuto come punto fisso dell'operatore, risolvendo l'equazione $\mathcal{T}_\mu V_\mu = V_\mu$.

La stessa equazione di punto fisso può essere riscritta in termini di **Q-factor** associati alla politica μ, ottenendo

$$Q_\mu(i,\mu(i)) = \mathbb{E}[h(i,\mu(i),j) + \gamma Q_\mu(j,\mu(j))],$$

dove l'aspettativa è presa rispetto allo stato successivo j. Astrattamente, il problema può essere ricondotto a un'equazione di punto fisso del tipo $y = Hy$, la cui soluzione tramite iterazione diretta $y^{(k)} = Hy^{(k-1)}$ non garantisce in generale la convergenza.

Anche nel caso in cui l'operatore H sia una contrazione, non è possibile valutarlo esattamente poiché la distribuzione di probabilità necessaria a calcolare l'aspettativa non è nota. Si riscrive allora l'equazione come $y = y + \alpha(Hy - y)$, da cui deriva lo schema iterativo $y^{(k)} = y^{(k-1)} + \alpha\big(Hy^{(k-1)} - y^{(k-1)}\big)$, che consente di applicare uno smoothing esponenziale sostituendo le aspettative con osservazioni casuali.

Applicando tale schema al contesto dei processi decisionali di Markov, si consideri di trovarsi allo stato $s^{(k)} = i$ e di applicare l'azione $a^{(k)} = \mu(i)$, osservando lo stato successivo $s^{(k+1)} = j$ e il contributo immediato $h(i, \mu(i), j)$. L'adattamento dello schema porta all'aggiornamento definito dalla **temporal difference**

$$\Delta^{(k)} = h(i, \mu(i), j) + \gamma \hat{Q}_\mu^{(k-1)}(j, \mu(j)) - \hat{Q}_\mu^{(k-1)}(i, \mu(i))$$

e dalla relazione

$$\hat{Q}_\mu^{(k)}(i, \mu(i)) = \hat{Q}_\mu^{(k-1)}(i, \mu(i)) + \alpha\, \Delta^{(k)}.$$

Il valore atteso della temporal difference sarebbe nullo se fossero noti i veri Q-factor; in pratica, si utilizza una singola osservazione casuale e una correzione non nulla indica la necessità di aggiornare la stima corrente, con un fattore di smoothing pari al learning rate α.

Poiché il metodo utilizza stime per costruire nuove stime, esso rientra nella classe dei metodi di **bootstrapping**. L'algoritmo così ottenuto prende il nome di **SARSA**, acronimo della sequenza (State, Action, Reward, State, Action), ed è un metodo **on-policy**, poiché il comportamento dell'agente segue la politica μ di cui si vuole apprendere il valore, come avviene nella policy iteration.

Per rendere operativo l'algoritmo sono necessari un adeguato livello di exploration, ad esempio tramite una strategia ε-greedy, e l'arresto anticipato dell'apprendimento per eseguire un passo di miglioramento della politica. Questo schema prende il nome di **generalized (o optimistic) policy iteration**, in quanto si assume in modo ottimistico di disporre di una valutazione sufficientemente accurata della politica corrente. Tale scelta può accelerare l'apprendimento, ma può avere effetti negativi sulla convergenza. Una volta ottenuta una stima $\hat{Q}_\mu(i, a)$ dei Q-factor, la politica viene aggiornata in modo greedy ponendo

$$\tilde{\mu}(i) \in \arg \operatorname*{opt}_{a \in A(i)} \hat{Q}_\mu(i, a);$$

a differenza dell'iterazione esatta, non vi è tuttavia garanzia che la nuova politica risulti effettivamente migliorata.

Q-learning per MDP finiti Il **Q-learning** è un approccio di reinforcement learning **model-free** alternativo per il trattamento di MDP finiti. Si richiamano innanzitutto le equazioni fondamentali della programmazione dinamica in termini di **Q-factor ottimi**, date da

$$Q(i, a) = \sum_{j \in S} \pi(i, a, j) \Big(h(i, a, j) + \gamma \operatorname*{opt}_{a' \in A(j)} Q(j, a') \Big), \qquad i \in S,\ a \in A(i),$$

e dalla relazione

$$V(i) = \operatorname*{opt}_{a \in A(i)} Q(i, a).$$

A differenza del caso SARSA, in cui si considerano Q-factor associati a una politica stazionaria fissata μ, in questo contesto si lavora direttamente con i fattori **ottimi**.

Poiché non si conoscono a priori né il contributo immediato $h(i, a, j)$ né le probabilità di transizione $\pi(i, a, j)$, tali quantità devono essere apprese tramite sperimentazione e campionamento. All'iterazione k, prima di selezionare l'azione $a^{(k)}$, si dispone di un insieme corrente di stime dei Q-factor, indicato con $\hat{Q}^{(k-1)}(i, a)$, dove l'indice $(k-1)$ segnala che le stime sono state aggiornate al passo precedente, dopo aver osservato l'effetto dell'azione $a^{(k-1)}$.

Trovandosi nello stato $s^{(k)} = i$, l'azione successiva viene selezionata sulla base delle stime correnti secondo una regola greedy,

$$a^{(k)} \in \arg \operatorname*{opt}_{a \in A(i)} \hat{Q}^{(k-1)}(i, a).$$

In questo senso, il Q-learning applica la stessa logica **off-policy** utilizzata nella value iteration. A differenza della policy iteration, non si valuta una politica fissata μ: la politica è implicita nei Q-factor e, poiché questi vengono aggiornati continuamente, la politica cambia nel tempo; in altre parole, si utilizza una politica per apprenderne un'altra.

Dopo aver applicato l'azione selezionata, si osservano lo stato successivo $s^{(k+1)} = j$ e il contributo immediato $h(i, a^{(k)}, j)$, oppure $f(i, a^{(k)})$ nel caso deterministico. Queste informazioni permettono di costruire una nuova osservazione di una variabile aleatoria che bilancia l'obiettivo di breve periodo e quello di lungo periodo, data da

$$\hat{q} = h\big(i, a^{(k)}, s^{(k+1)}\big) + \gamma \operatorname*{opt}_{a' \in A(s^{(k+1)})} \hat{Q}^{(k-1)}\big(s^{(k+1)}, a'\big),$$

che include sia l'osservazione casuale dello stato successivo sia il valore associato a tale stato seguendo la politica implicita nelle stime dei Q-factor al passo $k-1$, ossia

$$\hat{V}^{(k-1)}\big(s^{(k+1)}\big) = \operatorname*{opt}_{a' \in A(s^{(k+1)})} \hat{Q}^{(k-1)}\big(s^{(k+1)}, a'\big).$$

Il Q-factor relativo alla coppia stato–azione $\big(s^{(k)}, a^{(k)}\big)$ viene quindi aggiornato secondo la logica dello **smoothing esponenziale**,

$$\hat{Q}^{(k)}\big(s^{(k)}, a^{(k)}\big) = \alpha \hat{q} + (1-\alpha)\hat{Q}^{(k-1)}\big(s^{(k)}, a^{(k)}\big),$$

assumendo un coefficiente di smoothing costante α, sebbene sia possibile considerare anche coefficienti decrescenti. La stessa idea può essere espressa in termini di **temporal difference** definendo

$$\Delta^{(k)} = \Big[h\big(s^{(k)}, a^{(k)}, j\big) + \gamma \operatorname*{opt}_{a' \in A(j)} \hat{Q}^{(k-1)}(j, a')\Big] - \hat{Q}^{(k-1)}\big(s^{(k)}, a^{(k)}\big),$$

e scrivendo l'aggiornamento come

$$\hat{Q}^{(k)}(s, a) = \begin{cases} \hat{Q}^{(k-1)}(s, a) + \alpha\, \Delta^{(k)}, & \text{se } s = s^{(k)},\ a = a^{(k)}, \\ \hat{Q}^{(k-1)}(s, a), & \text{altrimenti,} \end{cases}$$

dove $j = s^{(k+1)}$ è lo stato successivo osservato.

La differenza tra SARSA e Q-learning emerge confrontando le correzioni introdotte nei due casi: entrambe aggiornano le stime dei valori stato–azione, ma SARSA è un metodo **on-policy**, riferito a una politica stazionaria μ fissata, mentre il Q-learning è **off-policy**, poiché la politica continua a cambiare durante il processo di apprendimento.

Implementazione MATLAB del Q-learning Una possibile implementazione del Q-learning in ambiente MATLAB è riportata nel codice mostrato, che traduce operativamente lo schema teorico discusso in precedenza. La funzione `QLearning` restituisce i Q-factor appresi, la politica stazionaria associata e una matrice che conta le visite delle coppie stato–azione, permettendo di monitorare il processo di apprendimento.

L'algoritmo interagisce con il sistema dinamico tramite un oggetto `systemObj`, che fornisce il contributo immediato e lo stato successivo in risposta a un'azione selezionata. A ogni iterazione l'azione viene scelta sulla base delle stime correnti dei Q-factor, con la possibilità di introdurre exploration mediante una strategia **ε-greedy**.

Dopo l'osservazione della transizione, il Q-factor corrispondente viene aggiornato tramite uno schema di **smoothing esponenziale**, combinando la nuova informazione con la stima precedente mediante il learning rate α. Al termine dell'apprendimento, la politica viene ricavata in modo greedy dai Q-factor, fornendo una realizzazione compatta e coerente dell'approccio **off-policy** del Q-learning per MDP finiti.

Definizione MATLAB di una classe di sistema generica Poiché il Q-learning è un approccio **model-free**, è necessario mantenere separato l'algoritmo di apprendimento dalla simulazione del sistema dinamico. Tale separazione è ottenuta tramite la classe MATLAB `systemClass`, che funge da interfaccia tra il Q-learner e il sistema da controllare, incapsulando stati, azioni ammissibili, probabilità di transizione e contributi immediati.

Il metodo centrale della classe è `getNext`, che, dato lo stato corrente e un'azione selezionata, restituisce il contributo immediato e lo stato successivo campionato secondo le probabilità di transizione, aggiornando lo stato interno dell'oggetto. In questo schema il contributo è associato alla coppia stato–azione, assumendo che sia deterministico oppure noto in valore atteso.

La struttura proposta fornisce un'interfaccia minimale ma sufficiente per l'integrazione con il Q-learning; in un'implementazione orientata agli oggetti più rigorosa, la classe dovrebbe essere astratta e limitarsi a definire l'interfaccia, demandando a classi derivate l'implementazione delle dinamiche specifiche.

Esempio numerico Dall'esempio numerico si evince che il comportamento del Q-learning dipende in modo critico sia dall'inizializzazione dei Q-factor sia dalla scelta del learning rate e della strategia di exploration. In condizioni favorevoli, con un'inizializzazione ragionevole e un numero sufficiente di iterazioni, le stime dei valori stato–azione risultano coerenti con i valori ottimi ottenuti tramite value iteration e sono sufficientemente accurate da indurre la politica ottima, anche quando alcune stime restano imprecise a causa di visite rare di certi stati. Tuttavia, inizializzazioni sfavorevoli possono portare a politiche subottimali persistenti, poiché azioni inizialmente penalizzate non vengono mai esplorate e i corrispondenti Q-factor non vengono aggiornati. Un aumento del learning rate può in parte mitigare il problema accelerando l'apprendimento, ma non è sufficiente quando l'esplorazione è completamente assente. L'esempio mostra infatti che, in mancanza di exploration, il Q-learning può rimanere intrappolato in politiche errate indipendentemente dal numero di iterazioni. L'introduzione di una strategia ε-greedy consente di superare tali difficoltà, garantendo che tutte le azioni vengano esplorate e permettendo il recupero della politica ottima anche a partire da inizializzazioni sfavorevoli, al prezzo però di un aumento significativo dello sforzo computazionale. Quindi, *l'esempio evidenzia il ruolo centrale del compromesso tra exploration ed exploitation per l'efficacia pratica del Q-learning.*

12 CAP 12 – Simulazione

12.1 Simulazione coda M/M/1 e intervalli di confidenza (code 1)

Contesto del problema Lo script simula una **coda M/M/1** (arrivi Poisson con tasso λ, tempi di servizio esponenziali con tasso μ, un solo server) per valutare la bontà degli **intervalli di confidenza** costruiti sui tempi di attesa tramite `normfit`.

Il sistema è **stabile** quando $\rho = \lambda/\mu < 1$. Nel codice si imposta $\lambda = 1$ e $\mu = 1.1$, e si stampa il **valore teorico** del tempo di attesa atteso, da usare come riferimento per confrontare gli intervalli ottenuti in simulazione.

L'esperimento viene ripetuto **10 volte** per osservare la variabilità degli intervalli e verificare se risultano **coerenti** tra loro (intervalli che si sovrappongono) e se includono il **vero valore**.

```
%% 1 - SIMULAZIONE CODA M/M/1
% Scopo: verificare la qualità degli intervalli di confidenza stimati via simulazione
% per il tempo di attesa medio in una coda M/M/1, usando normfit sui tempi di attesa.

rng 'default' % Inizializza il generatore di numeri casuali (riproducibilità)

inRate = 1;       % Tasso di arrivo: lambda
servRate = 1.1;   % Tasso di servizio: mu (stabilità se lambda/mu < 1)
                  % Nota: aumentando mu (es. 2.1) il sistema ha meno "memoria",
                  % la correlazione si riduce e gli intervalli risultano più affidabili.
```

```
% Valore teorico del tempo di attesa atteso (da confrontare con la simulazione)
fprintf(1,"Tempo di attesa atteso (teorico): %.4f\n\n",(inRate/servRate)/(servRate-inRate));

sampleSize = 600000; % Numero di clienti simulati per ciascuna replica

fprintf(1,"Intervalli di confidenza stimati:\n");
for k = 1:10 % Ripeto l'esperimento 10 volte per ottenere 10 intervalli di confidenza
    ww = MM1_Queue(inRate, servRate, sampleSize); % Tempi di attesa simulati (per tutti i
    clienti)

    % Stima di media e intervallo di confidenza assumendo (implicitamente) normalità e i.i.d.
    [~,~,ci] = normfit(ww);
    fprintf(1,"(%.4f, %.4f)\n",ci(1),ci(2));
end

% Funzione (annidata) per simulare la sequenza dei tempi di attesa in una M/M/1
function [waitTimes] = MM1_Queue(lambda, mu, howmany)
waitTimes = zeros(howmany,1); % Per convenzione, il primo cliente ha tempo di attesa 0

for j = 2:howmany
    % Tempi tra arrivi e tempi di servizio esponenziali
    intTime  = exprnd(1/lambda); % Inter-arrival time
    servTime = exprnd(1/mu);     % Service time

    % Ricorsione di Lindley:
    % il tempo di attesa del cliente j dipende dal tempo di attesa del cliente j-1
    waitTimes(j) = max(0, waitTimes(j-1) + servTime - intTime);
end
end

% Analisi diagnostiche sulla decima replica (ww dell'ultimo ciclo)

% Autocorrelazione: evidenzia dipendenza seriale nei tempi di attesa
figure;
autocorr(ww)

% Istogramma: controlla se la distribuzione empirica è compatibile con una normale
figure;
hist(ww)

% Media cumulata: verifica se/quanto la simulazione ha raggiunto il regime stazionario
figure;
t = (1:sampleSize)';
plot(t, cumsum(ww(t))./t)

% Osservazione: se gli intervalli non si sovrappongono, l'inferenza è instabile.
% Possibili cause principali:
% 1) I tempi di attesa non sono indipendenti: la correlazione porta a sottostimare la varianza.
% 2) La normalità non è una buona approssimazione per ww.
% 3) La stima può essere influenzata dal transitorio (warm-up) e non riflettere pienamente il
    regime.
```

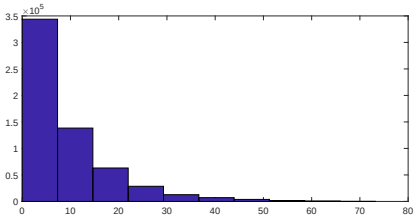
Conclusioni Dalla simulazione emergono tre criticità legate all'uso diretto di `normfit(ww)` sui tempi di attesa, e i tre plot le rendono **molto evidenti**.

(i) Dipendenza seriale. Il grafico di autocorrelazione mostra una **correlazione molto alta** anche a lag elevati: i tempi di attesa non sono i.i.d., perché derivano dalla ricorsione di Lindley. Questo significa che `normfit` lavora con un'ipotesi strutturalmente sbagliata e tende a produrre intervalli **troppo ottimisti** (varianza sottostimata rispetto alla varianza "effettiva" con correlazione).

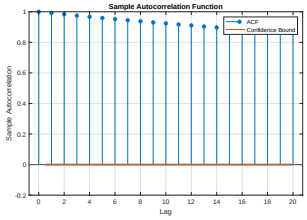
(ii) Ipotesi di normalità non adeguata. L'istogramma dei tempi di attesa è fortemente asimmetrico (massa concentrata su valori piccoli e **coda lunga**): è difficile giustificare un fit normale per la distribuzione di `ww`. Di conseguenza, anche quando gli intervalli numericamente sembrano ragionevoli, l'inferenza "tipo normale" può risultare fuorviante.

(iii) Effetti di transitorio. La media cumulata cresce rapidamente all'inizio e poi si assesta lentamente verso un valore stabile: questo indica un **warm-up non trascurabile**. Se non si gestisce il transitorio (ad es. eliminando i primi campioni), la stima della media e gli intervalli risultano influenzati dalla fase iniziale e possono non riflettere correttamente il regime stazionario.

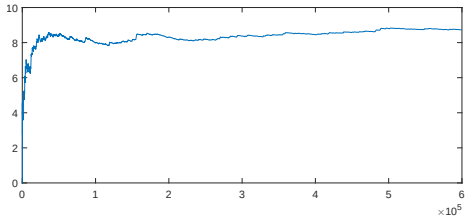
In sintesi, i plot mostrano che **correlazione elevata**, **non-normalità** e **transitorio** sono presenti contemporaneamente: per questo gli intervalli prodotti da `normfit(ww)` non sono un indicatore affidabile della precisione della stima, anche quando il valore teorico appare "vicino" ai risultati osservati.



(a) Istogramma dei tempi di attesa



(b) Autocorrelazione dei tempi di attesa



(c) Media cumulata dei tempi di attesa

Figure 1: Diagnostiche sulla simulazione della coda M/M/1: distribuzione empirica, dipendenza seriale e convergenza al regime stazionario.

12.2 Simulazione (s,S) con revisione periodica: script *extremely naive* (code 2)

Contesto del problema Lo script implementa una simulazione **molto semplificata** di una politica di controllo scorte **(s,S) a revisione periodica**.

L'obiettivo è capire se la politica sia **"buona"** oppure no; tuttavia, in un problema di inventory, il concetto di bontà non è automatico: serve scegliere un *criterio di valutazione* (ad esempio livello di servizio, vendite perse, costo di mantenimento, costo totale, ecc.).

Nel modello si osserva il sistema su un orizzonte discreto T e, a ogni periodo, si aggiorna il livello di scorta disponibile, si soddisfa (se possibile) la domanda e si decide se emettere un ordine: se la scorta finale scende sotto la soglia s, si ordina quanto basta per riportarsi a S.

```
%% 2 - EXTREMELY NAIVE: simulazione politica (s,S) a revisione periodica
% Scopo: simulare una politica (s,S) e valutare (in modo qualitativo) se la scelta
% dei parametri è soddisfacente. Per giudicare "buona/cattiva" una policy serve però
% fissare una metrica (es. vendite perse, stockout, livello di servizio, costi).

% Parametri della policy (s,S)
smallS = 50;   % soglia di riordino: se l'inventario finale <= s allora riordino
bigS   = 100;  % livello target: dopo il riordino voglio tornare a S

% Parametri della domanda (assunzione: domanda ~ Normale)
mu    = 60;
sigma = 10;

% Orizzonte di simulazione e vettori di output
```

```
T = 50;                  % orizzonte temporale discreto
onHand    = zeros(T,1);  % scorta disponibile a inizio periodo (dopo arrivi)
aveOnHand = zeros(T,1);  % scorta media nel periodo (proxy per holding cost)
lostSales = zeros(T,1);  % vendite perse (assunzione: no backorder)
stockOut  = NaN(T,1);    % indicatore stockout (true/false)
ordered   = zeros(T,1);  % quantità ordinata a fine periodo

% Scenario di domanda (caso semplice per debug)
demand = repmat(10,T,1); % domanda costante = 10 per tutti i periodi

% Alternativa: domanda casuale normale (attenzione a domande negative se sigma è grande)
% demand = normrnd(mu,sigma,T,1);

% Stato iniziale del sistema
endInv  = bigS;  % inventario finale del periodo precedente (inizializzato a S)
onOrder = 0;     % ordine in arrivo dal periodo precedente (lead time = 1 periodo)

% Simulazione a tempo discreto (revisione periodica)
for t = 1:T
    % Arrivo dell'ordine emesso nel periodo precedente
    onHand(t) = endInv + onOrder;

    % Soddisfazione della domanda nel periodo t
    if demand(t) <= onHand(t)
        % nessuno stockout: domanda interamente soddisfatta
        stockOut(t)  = false;
        lostSales(t) = 0;
        endInv       = onHand(t) - demand(t);
    else
        % stockout: non posso soddisfare tutta la domanda
        stockOut(t)  = true;
        lostSales(t) = demand(t) - onHand(t);
        endInv       = 0;
    end

    % Scorta media nel periodo (utile per stimare un costo di holding)
    aveOnHand(t) = (onHand(t) + endInv) / 2;

    % Decisione di riordino a fine periodo secondo politica (s,S)
    if endInv <= smallS
        % sottosoglia: ordino per riportarmi a S
        onOrder    = bigS - endInv;
        ordered(t) = onOrder;
    else
        % sopra soglia: non ordino
        onOrder    = 0;
        ordered(t) = 0;
    end
end
```

Conclusioni Lo script produce una **traiettoria campionaria completa** del sistema di inventario sotto una politica **(s,S) a revisione periodica**, ma il suo output rimane **puramente descrittivo** e non immediatamente interpretabile in termini di performance.

(i) Esistenza di un output, ma non di una valutazione. La simulazione genera diverse serie tem-

porali di interesse: livello di scorta disponibile (`onHand`), scorta media per periodo (`aveOnHand`), vendite perse (`lostSales`), indicatori di stockout (`stockOut`) e quantità ordinate (`ordered`). Tuttavia, nessuna di queste variabili viene aggregata in un indicatore sintetico che permetta di giudicare la bontà della policy.

(ii) Interpretazione dei risultati con domanda costante. Nel caso mostrato (`demand = 10`), la domanda è molto inferiore al livello target $S = 100$, per cui il sistema tende rapidamente a stabilizzarsi: non si osservano stockout e le vendite perse sono nulle. Questo comportamento è coerente, ma poco informativo, perché non mette sotto stress la politica (s,S).

In sintesi, lo script ha valore didattico perché chiarisce la **meccanica della politica (s,S)** e la sua implementazione in tempo discreto, ma per diventare uno strumento decisionale deve essere esteso introducendo una **funzione obiettivo** (ad esempio costo totale o livello di servizio medio) e scenari di domanda più realistici.

12.3 Politica (s,S) a revisione periodica: simulazione su scenari e ottimizzazione dei parametri (code 3a, 3b, 3c)

Contesto del problema Questa parte unifica tre blocchi coerenti: **(3a)** una funzione che simula una politica **(s,S)** su uno o più scenari di domanda, **(3b)** uno script di **ottimizzazione** dei parametri (s, S) minimizzando il **costo medio** stimato via simulazione, e **(3c)** un test su domanda costante per confrontare i risultati con un riferimento tipo **EOQ**.

L'impostazione è quella tipica della **simulation–optimization**: la funzione obiettivo non è disponibile in forma chiusa, ma viene **stimata** (costo totale) tramite simulazione; in seguito, la stima viene minimizzata con algoritmi di ricerca che non richiedono derivate (qui `patternsearch` e `surrogateopt`).

Gli scenari di domanda vengono generati **all'esterno** della funzione per garantire modularità e confronti ripetibili tra soluzioni. Inoltre si distingue tra **in-sample** (usato per ottimizzare) e **out-of-sample** (utile per controllare la robustezza, anche se qui non impiegato direttamente).

Nota operativa. `SS_Simulation` è una **funzione**: se viene eseguita direttamente dal Command Window senza argomenti (es. `» SS_Simulation`), MATLAB genera l'errore *Not enough input arguments*. La funzione è pensata per essere chiamata da (3b–3c) tramite `objfun`, passando esplicitamente tutti gli input.

```
%% 3a - Simulation of (s,S) periodic review policy (funzione)
% Scopo: simulare una policy (s,S) su più scenari di domanda, calcolando:
% - totalCost(k): costo totale della replica k (da usare in ottimizzazione)
% - freqStockout(k): frequenza di periodi con vendite perse (proxy servizio)

function [totalCost, freqStockout] = SS_Simulation(smallS, bigS, demandScenarios, ...
    initialStock, onHandCost, lostPenalty, unitCost, fixedCharge)

% Parametri della policy
% smallS, bigS    : soglia s e livello target S (con S >= s)
% demandScenarios : matrice scenari (righe = repliche, colonne = periodi)
% initialStock    : scorta iniziale (può influenzare il transitorio)

% Parametri economici
% onHandCost   : costo di giacenza per unità media in stock
% lostPenalty  : penalità per unità di domanda persa (vendite perse)
% unitCost     : costo unitario per unità ordinata
% fixedCharge  : costo fisso ogni volta che si effettua un ordine (setup)

% Nota: ogni riga di demandScenarios è una replica di durata T
[numScenarios, T] = size(demandScenarios);

% Output: un valore per ciascuno scenario
totalCost    = zeros(numScenarios,1);
```

```
freqStockout = zeros(numScenarios,1);

% Vettori di monitoraggio per una singola replica
onHand    = zeros(T,1);    % scorta disponibile a inizio periodo
aveOnHand = zeros(T,1);    % scorta media nel periodo
lostSales = zeros(T,1);    % vendite perse
stockOut  = NaN(T,1);      % true/false stockout
ordered   = zeros(T,1);    % quantità ordinata a fine periodo

% Ciclo sulle repliche (scenari)
for k = 1:numScenarios

    % Stato iniziale della replica k
    endInv  = initialStock;  % scorta finale iniziale
    onOrder = 0;             % nulla in transito (lead time = 1 periodo)
    demand  = demandScenarios(k,:);

    % Evoluzione temporale (revisione periodica)
    for t = 1:T
        % Arrivo dell'ordine emesso nel periodo precedente
        onHand(t) = endInv + onOrder;

        % Soddisfazione domanda (assunzione: vendite perse, no backorder)
        if demand(t) <= onHand(t)
            stockOut(t)  = false;
            lostSales(t) = 0;
            endInv       = onHand(t) - demand(t);
        else
            stockOut(t)  = true;
            lostSales(t) = demand(t) - onHand(t);
            endInv       = 0;
        end

        % Scorta media del periodo (per holding cost)
        aveOnHand(t) = (onHand(t) + endInv)/2;

        % Decisione di riordino secondo (s,S)
        if endInv <= smallS
            onOrder    = bigS - endInv;  % riordino per tornare a S
            ordered(t) = onOrder;
        else
            onOrder    = 0;
            ordered(t) = 0;
        end
    end

    % Costo totale della replica k
    totalCost(k) = onHandCost*sum(aveOnHand) + lostPenalty*sum(lostSales) + ...
        unitCost*sum(ordered) + fixedCharge*sum(ordered>0);

    % Frequenza stockout: frazione di periodi con lostSales > 0
    freqStockout(k) = sum(lostSales>0)/T;
end
```

```
%% 3b - Optimization: Script to optimize (s,S) parameters
```

```
% Economics
unitCost = 10;
onHandCost = 0.1*unitCost;
lostPenalty = 2*unitCost; % penalità elevata per vendite perse
fixedCharge = 1000;

% Demand distribution (oggetto distribuzione)
pd = makedist('NegativeBinomial','R',20,'P',.3);

% Make inSample scenario
rng default
horizonInSample = 30000; % una replica molto lunga
inSampleDemand = random(pd,1,horizonInSample);

% Initial stock (scelta per ridurre l'effetto del transitorio)
initialStock = mean(pd)+2*std(pd);

% Objective function (minimizza costo medio simulato)
% Parametrizzazione: x(1)=s, x(2)=S-s così impongo S>=s
objfun = @(x) mean(SS_Simulation(x(1), x(1)+x(2), inSampleDemand, ...
    initialStock, onHandCost, lostPenalty, unitCost, fixedCharge));

upperBounds = [mean(pd)+10*std(pd);50*mean(pd)];

% Use pattern search
xStar = patternsearch(objfun,[2*mean(pd);5*mean(pd)],[],[],[],[],zeros(2,1),upperBounds);
smallS_Pattern = xStar(1);
bigS_Pattern = xStar(1)+xStar(2);
fprintf(1,'Pattern Search: smallS = %.2f, bigS = %.2f\n', smallS_Pattern, bigS_Pattern);

% Use surrogate opt
xStar = surrogateopt(objfun,zeros(2,1),upperBounds);
smallS_Surrogate = xStar(1);
bigS_Surrogate = xStar(1)+xStar(2);
fprintf(1,'Surrogate Opt: smallS = %.2f, bigS = %.2f\n', smallS_Surrogate, bigS_Surrogate);
```

```
%% 3c - Test EOQ Script to optimize (s,S) parameters (domanda costante)

% Economics
unitCost = 10;
onHandCost = 0.1*unitCost;
lostPenalty = 2*unitCost;
fixedCharge = 10000;

% Make inSample scenario
rng default
horizonInSample = 80000;
D = 5; % domanda costante
inSampleDemand = D*ones(1,horizonInSample);

EOQ = sqrt(2*D*fixedCharge/onHandCost)
initialStock = EOQ;

% Create objfun
```

```
objfun = @(x) mean(SS_Simulation(x(1), x(1)+x(2), inSampleDemand, ...
    initialStock, onHandCost, lostPenalty, unitCost, fixedCharge));
upperBounds = [100*initialStock;100*initialStock];

% Use pattern search
xStar = patternsearch(objfun,upperBounds,[],[],[],[],zeros(2,1),upperBounds);

smallS_Pattern = xStar(1);
bigS_Pattern = xStar(1)+xStar(2);
fprintf(1,'Pattern Search: smallS = %.2f, bigS = %.2f\n', smallS_Pattern, bigS_Pattern);
```

Conclusioni Il blocco (3a–3c) realizza un flusso completo di **simulation–optimization** per una policy (s,S).

(i) Output della simulazione. La funzione `SS_Simulation` produce due output utili: **costo totale** (vettore per replica) e **frequenza di stockout**. In (3b–3c) viene ottimizzata la **media** del costo totale, quindi l'output è direttamente trasformato in una funzione obiettivo minimizzabile.

(ii) Interpretazione della figura di (3b). Il grafico di `patternsearch` mostra la riduzione del **best function value** lungo le iterazioni fino a stabilizzarsi: è l'evidenza che l'algoritmo sta esplorando lo spazio dei parametri e convergendo verso un valore di costo più basso. Coerentemente, nel Command Window vengono stampate le coppie (s, S) trovate (Pattern Search e Surrogate Opt).

(iii) Interpretazione del test (3c). Nel caso a domanda costante, `EOQ` fornisce una scala di riferimento per l'ordine "ragionevole". L'ottimizzazione via `patternsearch` porta infatti a un S dell'ordine di grandezza dell'EOQ (come si vede dall'output stampato), mentre s può risultare molto piccolo: con domanda deterministica e costo fisso elevato, la logica è **riordinare raramente** ma in lotti grandi, riducendo il numero di ordini.

In sintesi, (3a) definisce una metrica quantitativa, (3b) la minimizza su uno scenario stocastico in-sample e (3c) controlla che la soluzione sia coerente in un caso deterministico benchmark.

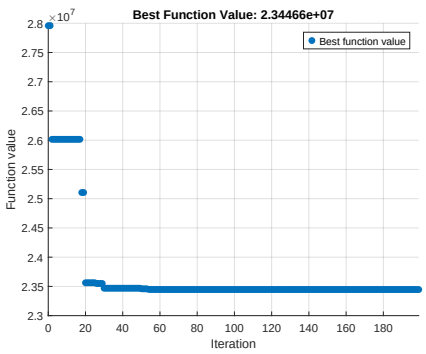


Figure 2: Andamento del *best function value* durante l'ottimizzazione dei parametri (s, S) tramite `patternsearch` nello scenario stocastico in-sample.

12.4 Coda M/M/1 con balking (FIFO): simulazione percentuale di clienti che rinunciano (code 4a, 4b)

Contesto del problema Si considera una **coda M/M/1** (interarrivi e tempi di servizio esponenziali, tassi assegnati) con disciplina **FIFO** e un meccanismo di **balking**: quando un cliente arriva, decide se entrare o rinunciare in funzione della lunghezza della coda.

La regola di balking è la seguente: il cliente **entra sicuramente** se la coda è lunga meno di 10 clienti, entra con **probabilità 50%** se la lunghezza è tra 10 e 15 (estremi inclusi), e **rinuncia sicuramente** se la coda è maggiore di 15.

L'obiettivo della simulazione è stimare, su un numero fissato di clienti serviti, la **percentuale di clienti che rinunciano** (ossia che non si mettono in coda).

Il codice propone due implementazioni: **(4a)** una versione script "diretta" basata sulla simulazione a eventi discreti (arrivo/completamento), e **(4b)** una versione più pulita in forma di funzione, che incapsula lo **stato del sistema** e separa la logica degli eventi tramite funzioni annidate.

```
%% 4a - ESAME BA 03/07/2024
% Coda M/M/1 FIFO con balking: stimo quanti clienti rinunciano a entrare.
% Regola di balking:
% - se la lunghezza della coda e' < 10: il cliente entra sempre
% - se la lunghezza e' tra 10 e 15 (inclusi): entra con probabilita' 50%
% - se la lunghezza e' > 15: rinuncia sicuramente

%% VERSIONE 1 - Script M/M/1 con balking

% Tassi
arrivalRate = 1;
serviceRate = 1.4;

% Stato del sistema
numInQueue = 0; % numero nel sistema (incluso l'eventuale cliente in servizio)
count = 0;      % numero di completamenti di servizio
lost  = 0;      % numero di clienti persi per balking

% Tempi del prossimo evento (simulazione a eventi discreti)
nextCompletion = inf;                 % sistema inizialmente vuoto
nextArrival    = exprnd(1/arrivalRate);
toServe = 10000;                      % voglio servire questo numero di clienti

while count <= toServe

    % Evento di completamento: avviene se e' il prossimo tra i due eventi
    if nextCompletion <= nextArrival
        clock = nextCompletion;       % avanzo l'orologio
        count = count + 1;            % ho completato un servizio

        if numInQueue == 0
            % nessuno in coda: il server diventa idle == il server è libero, non sta servendo
   nessuno
            nextCompletion = inf;
        else
            % qualcuno in coda: parte subito il servizio del prossimo cliente
            numInQueue = numInQueue - 1;
            nextCompletion = clock + exprnd(1/serviceRate);
        end

    % Evento di arrivo
    else
        clock = nextArrival;          % avanzo l'orologio

        if numInQueue == 0
            % server idle: il cliente entra e inizia subito il servizio
            numInQueue = 1;
            nextCompletion = clock + exprnd(1/serviceRate);

        elseif numInQueue < 10
            % entra sicuramente
            numInQueue = numInQueue + 1;
```

```
        elseif numInQueue <= 15
            % entra con probabilita' 50% (altrimenti balking)
            if rand > 0.5
                lost = lost + 1;
            else
                numInQueue = numInQueue + 1;
            end

        else
            % coda troppo lunga: balking certo
            lost = lost + 1;
        end

        % Pianifico il prossimo arrivo
        nextArrival = clock + exprnd(1/arrivalRate);
    end
end

fprintf(1,'%d customers balked\n', lost);
```

```
%% 4b - VERSIONE 2: funzione M/M/1 con balking (piu' modulare)
% Idea: incapsulo lo stato in una struct e separo la logica dei due eventi
% (completamento e arrivo) tramite funzioni annidate.

function lost = MM1_Balking(arrivalRate,serviceRate,toServe)

% Stato del sistema (incapsulato)
state.numInQueue = 0;   % numero nel sistema (include il cliente in servizio)
state.count      = 0;   % numero di completamenti
state.lost       = 0;   % clienti persi per balking
state.clock      = 0;   % tempo corrente

% Tempi del prossimo evento
state.nextCompletion = inf;
state.nextArrival    = exprnd(1/arrivalRate);

% Simulazione a eventi: ad ogni passo scelgo l'evento piu' vicino
while state.count <= toServe
    if state.nextCompletion <= state.nextArrival
        state.clock = state.nextCompletion;
        state = manageService(state);
    else
        state.clock = state.nextArrival;
        state = manageArrival(state);
    end
end

lost = state.lost;

% --- Funzioni annidate: aggiornano lo stato dato un evento ---

    function state = manageService(state)
        % Gestione completamento di servizio
        state.count = state.count + 1;
```

```
        if state.numInQueue == 0
            % nessuno in coda: server idle
            state.nextCompletion = inf;
        else
            % prendo il prossimo cliente in coda e schedulo il completamento
            state.numInQueue = state.numInQueue - 1;
            state.nextCompletion = state.clock + exprnd(1/serviceRate);
        end
    end

    function state = manageArrival(state)
        % Gestione arrivo con regola di balking
        if state.numInQueue == 0
            % server idle: servizio immediato
            state.numInQueue = 1;
            state.nextCompletion = state.clock + exprnd(1/serviceRate);

        elseif state.numInQueue < 10
            % entra sicuramente
            state.numInQueue = state.numInQueue + 1;

        elseif state.numInQueue <= 15
            % entra con probabilita' 50%
            if rand > 0.5
                state.lost = state.lost + 1;
            else
                state.numInQueue = state.numInQueue + 1;
            end

        else
            % balking certo
            state.lost = state.lost + 1;
        end

        % Pianifico il prossimo arrivo
        state.nextArrival = state.clock + exprnd(1/arrivalRate);
    end

end
```

Conclusioni Il codice implementa correttamente una simulazione a **eventi discreti** con due soli eventi possibili (arrivo/completamento), includendo una regola di balking dipendente dalla lunghezza della coda.

(i) Output di interesse. La quantità `lost` rappresenta il numero di clienti che rinunciano; per ottenere la **percentuale** richiesta basta normalizzare rispetto agli arrivi totali simulati (oppure rispetto a `lost + served`, a seconda della definizione adottata).

(ii) Struttura a eventi e stato. La versione (4b) è più pulita perché incapsula lo stato e separa la logica dei due eventi, rendendo più facile estendere il modello (ad esempio aggiungendo nuove regole o nuove metriche).

(iii) Interpretazione del parametro di soglia. Le soglie 10 e 15 regolano direttamente il trade-off tra congestione e perdita di domanda: soglie più basse riducono la congestione ma aumentano i clienti persi, soglie più alte fanno l'opposto.

In sintesi, la versione (4b) è preferibile per riuso e manutenzione, mentre la versione (4a) è utile come implementazione immediata e trasparente della logica di simulazione.

12.5 Two-stage ATP/ATO: modello ai valori attesi vs modello stocastico (code 5a,5b)

Contesto del problema Questi script confrontano due approcci per un problema di pianificazione in presenza di domanda incerta, formulato come **problema a due stadi**.

Nel **primo stadio** si decide quante componenti produrre (`make`), soggette a vincoli di capacità sulle risorse. Nel **secondo stadio** si decide quante unità di prodotto finito vendere (`sell`) una volta osservata la domanda (che varia per scenario).

Il codice implementa due strategie: (i) un **modello deterministico** basato sulla **domanda media** (Expected Value, EV), e (ii) un **modello stocastico** (Recourse Problem, RP) che ottimizza il profitto atteso su più scenari.

Nel blocco 5b, oltre alla soluzione EV e RP, viene calcolata la metrica **VSS** (*Value of the Stochastic Solution*), che quantifica il guadagno di usare il modello stocastico rispetto a "fissare" le decisioni di primo stadio ottenute dal modello ai valori attesi.

```
%% 5a - Two Stages ATP (con intlinprog): modello ai valori attesi vs modello stocastico

% DATI DEL PROBLEMA
numItems     = 3;
numComp      = 5;
numScenarios = 3;
numResources = 3;

compCost = [20; 30; 10; 10; 10];   % costo componenti (primo stadio)
gozinto  = [ 1 1 1 0 0
             1 1 0 1 0
             1 1 0 0 1 ];          % matrice di distinta base (prodotti -> componenti)

needRes  = [ 1 2 1
             1 2 2
             2 2 0
             1 2 0
             3 2 0 ];              % consumo risorse per ogni componente

availRes = [800; 700; 600];        % disponibilita' risorse
endPrice = [80; 70; 90];           % prezzo di vendita dei prodotti

demand = [ 100  50 120
            50  25  60
           100 110  60 ];          % domanda per scenario (righe=prodotti, colonne=scenari)

probs = ones(numScenarios,1)/numScenarios;
demandAverage = mean(demand,2);

%% 1) MODELLO DETERMINISTICO AI VALORI ATTESI (EV) - variabili intere
% Variabili di decisione: [make; sell]
% Obiettivo: massimizzare profitto = ricavi - costi
% In intlinprog si minimizza, quindi cambio segno ai ricavi ( -endPrice )

objCoeff = [compCost; -endPrice];

% Bounds: componenti senza upper bound, vendite limitate dalla domanda media
lb = zeros(numComp+numItems,1);
ub = [repmat(Inf,numComp,1); demandAverage];
```

```
% Vincoli componenti: gozinto' * sell <= make
A1 = [-eye(numComp), gozinto'];
b1 = zeros(numComp,1);

% Vincoli risorse: needRes' * make <= availRes
A2 = [needRes', zeros(numResources,numItems)];
b2 = availRes;

A = [A1; A2];
b = [b1; b2];

% Risoluzione
[xVar, aux] = intlinprog(objCoeff,1:length(objCoeff),A,b,[],[],lb,ub);
profit  = -aux;
display(xVar);
display(profit);

%% 2) MODELLO STOCASTICO (RP) - variabili continue (secondo stadio per scenario)
% Variabili: [make; sell_1; ...; sell_S]
% Obiettivo: costi primo stadio + ricavi attesi sui vari scenari

objCoeff = [compCost];
for s = 1:numScenarios
    objCoeff = [objCoeff; -probs(s)*endPrice];
end

lb = zeros(numComp+numScenarios*numItems,1);
ub = [repmat(Inf,numComp,1); reshape(demand,numItems*numScenarios,1)];

% Matrice vincoli con struttura a blocchi (due stadi)
A = zeros(numResources+numScenarios*numComp, numComp+numScenarios*numItems);

% Vincoli risorse (uguali per tutti gli scenari)
A(1:numResources, 1:numComp) = needRes';

% Vincoli componenti per ciascuno scenario: gozinto' * sell_s <= make
baseRow = numResources;
baseCol = numComp;
for s = 1:numScenarios
    A(baseRow+(1:numComp), 1:numComp) = -eye(numComp);
    A(baseRow+(1:numComp), baseCol+(1:numItems)) = gozinto';
    baseRow = baseRow + numComp;
    baseCol = baseCol + numItems;
end

b = [availRes; zeros(numScenarios*numComp,1)];

% Risoluzione
[xVarS, aux] = intlinprog(objCoeff,1:length(objCoeff),A,b,[],[],lb,ub);
profitS = -aux;
display(xVarS);
display(profitS);
```

```
%% 5b - In Two Stage ATO (con optimproblem): EV vs RP e calcolo VSS
```

```
format bank
typeVar = 'integer';      % soluzione con variabili intere
% typeVar = 'continuous'; % alternativa: rilassamento continuo

% DATI DEL PROBLEMA (stessi dati di 5a)
numItems     = 3;
numComp      = 5;
numScenarios = 3;
numResources = 3;

compCost = [20; 30; 10; 10; 10];
gozinto  = [ 1 1 1 0 0
             1 1 0 1 0
             1 1 0 0 1 ];
needRes  = [ 1 2 1
             1 2 2
             2 2 0
             1 2 0
             3 2 0 ];
availRes = [800; 700; 600];
endPrice = [80; 70; 90];

demand = [ 100  50 120
            50  25  60
           100 110  60 ];
probs = ones(numScenarios,1)/numScenarios;

%% 1) PROBLEMA AI VALORI ATTESI (EV)
demandAverage = mean(demand,2);

probEV = optimproblem('ObjectiveSense','max');
x = optimvar('x', numComp, 1, 'LowerBound', 0, 'Type', typeVar);                  % componenti
y = optimvar('y', numItems, 1, 'LowerBound', 0, 'UpperBound', demandAverage, ...
             'Type', typeVar);                                                    % prodotti
    finiti (media)

probEV.Objective = -dot(compCost,x) + dot(endPrice,y);
probEV.Constraints.cons1 = needRes' * x <= availRes;     % vincoli capacita'
probEV.Constraints.cons2 = gozinto' * y <= x;            % vincolo distinta base

showproblem(probEV)
[solEV, profitEV] = solve(probEV);

solEV.x
solEV.y
profitEV

%% 2) PROBLEMA STOCASTICO (RP) con ricorso
probRP = optimproblem('ObjectiveSense','max');

x = optimvar('x', numComp, 1, 'LowerBound', 0, 'Type', typeVar);                  % primo stadio
y = optimvar('y', numItems, numScenarios, 'LowerBound', 0, 'UpperBound', demand, ...
             'Type', typeVar);                                                    % secondo
    stadio per scenario
```

```
% Ricavo atteso: per ogni scenario calcolo endPrice'*y(:,s) e faccio media pesata
probRP.Objective = -dot(compCost,x) + dot(endPrice'*y, probs);

probRP.Constraints.cons1 = needRes' * x <= availRes;     % vincoli primo stadio

% Vincoli di componente per ogni scenario: gozinto' * y(:,s) <= x
compConstr = optimconstr(numScenarios,numComp);
for s = 1:numScenarios
    compConstr(s,:) = gozinto' * y(:,s) <= x;
end
probRP.Constraints.cons2 = compConstr;

showproblem(probRP)
[solRP, profitRP] = solve(probRP);

solRP.x
solRP.y
profitRP

%% 3) VALUTAZIONE DELLA SOLUZIONE EV NEL MODELLO STOCASTICO (calcolo VSS)
% Fisso la decisione di primo stadio alla soluzione EV e ricalcolo il profitto
probRP.Constraints.fixFirstStage = x == solEV.x;
[solRP2, profitRP2] = solve(probRP);

solRP2.y
profitRP2

% VSS: valore aggiunto della soluzione stocastica rispetto alla EV "valutata" su scenari
VSS = profitRP - profitRP2
format
```

Conclusioni I blocchi (5a–5b) mostrano chiaramente la differenza tra un modello ai **valori attesi** (EV) e un modello **stocastico a due stadi** (RP), e quantificano il beneficio di considerare l'incertezza tramite la metrica **VSS**.

(i) Soluzione EV (domanda media). Nel caso EV, il solver trova una soluzione ottima con profitto pari a **3220**. La decisione di primo stadio risulta `x` = (116,116,26,0,90) e la decisione di vendita sulla domanda media è `y` = (26,0,90). Questo conferma che il modello EV ottimizza rispetto a una domanda "aggregata", ignorando esplicitamente la variabilità tra scenari.

(ii) Soluzione stocastica RP. Nel modello a due stadi (RP), la soluzione ottima produce un profitto atteso pari a **2883.33**, con decisione di primo stadio `x` = (115,115,55,0,65). Al secondo stadio, le vendite `y` dipendono dallo scenario e si adattano ai diversi vincoli di domanda: questo è esattamente l'effetto del **ricorso** (si decide cosa vendere dopo aver osservato lo scenario).

(iii) Valutazione della soluzione EV nel modello stocastico e VSS. Fissando nel problema stocastico la decisione di primo stadio uguale a quella EV (`x == solEV.x`), il profitto ottenuto scende a **2320**. La differenza tra la soluzione pienamente stocastica e la soluzione EV "trapiantata" nel modello a scenari è la **VSS**, che nei risultati vale: VSS = 2883.33 − 2320 = **563.33**. Questo valore misura il guadagno ottenibile scegliendo **direttamente** una soluzione ottimizzata per scenari, anziché ottimizzare su domanda media e poi subire le conseguenze della variabilità.

In sintesi, anche se il profitto EV (**3220**) è più alto nel suo modello deterministico "ideale", quando si passa a valutare su scenari la soluzione EV perde robustezza (profitto **2320**), mentre la soluzione RP è costruita per gestire l'incertezza e garantisce un profitto atteso maggiore in quel contesto (**2883.33**), con un vantaggio quantificato da **VSS = 563.33**.

12.6 Capacity control e tariffe aeree: protezione della classe alta e simulazione degli arrivi (code 6)

Contesto del problema Lo script modella un problema di **revenue management** per un volo con capacità limitata: esistono due classi tariffarie con prezzi diversi (classe 1 più costosa, classe 2 più economica) e le richieste arrivano nel tempo secondo processi di Poisson.

L'obiettivo è applicare una regola semplice di **protezione** (booking limit): si riserva un certo numero di posti alla classe più remunerativa, rifiutando parte delle richieste della classe 2 quando il livello di protezione è stato raggiunto.

Il punto chiave è che la decisione viene presa usando tassi di arrivo **stimati** $(\hat{\lambda}_1, \hat{\lambda}_2)$, mentre la simulazione degli arrivi avviene usando i tassi **veri** (λ_1, λ_2): in questo modo si evidenzia l'impatto dell'errore di stima sul profitto.

Il **livello di protezione** viene calcolato tramite un quantile Poisson: si usa `poissinv` sul numero atteso di arrivi della classe 1 nell'intero orizzonte T.

```
%% 6 - Aircraft capacity and tariffs
% Scopo: gestire la capacita' (posti) di un volo con due classi tariffarie.
% Applico una regola di protezione per la classe alta: accetto la classe 2
% solo finche' non "invado" i posti riservati alla classe 1.

% Dati del problema
cap    = 100;   % capacita' totale (posti)
price1 = 400;   % prezzo classe 1 (alta)
price2 = 200;   % prezzo classe 2 (bassa)

% Orizzonte temporale (giorni fino alla partenza)
T = 60;

% Tassi di arrivo (per giorno): veri e stimati
trueArrivalRate1 = 0.4;  % tasso vero classe 1
trueArrivalRate2 = 2.2;  % tasso vero classe 2

hatArrivalRate1  = 0.5;  % tasso stimato classe 1 (usato per decidere)
hatArrivalRate2  = 2;    % tasso stimato classe 2 (qui non entra nella regola)

% Calcolo del livello di protezione (booking limit)
% Nota: il numero di richieste classe 1 sull'orizzonte T e' Poisson con media hatArrivalRate1*T
    .
% Uso un quantile: protezioneLevel = quantile_{1 - price2/price1}(Poisson(hatArrivalRate1*T))
protectionLevel = poissinv(1 - price2/price1, hatArrivalRate1*T);

% Stato del sistema
count1 = 0;   % numero di richieste accettate per classe 1
count2 = 0;   % numero di richieste accettate per classe 2
full   = false;

% Tempi del prossimo arrivo per le due classi (simulati sui tassi veri)
nextArrival1 = exprnd(1/trueArrivalRate1);
nextArrival2 = exprnd(1/trueArrivalRate2);

% Condizione di stop: nessun arrivo prima della partenza
stop = min(nextArrival1, nextArrival2) > T;

% Simulazione a eventi discreti: arrivi di classe 1 e classe 2
% Il loop termina quando (i) non arrivano piu' richieste prima della partenza oppure (ii) l'
```

```
    aereo e' pieno
while ~stop && ~full

    if nextArrival1 < nextArrival2
        % Prossimo evento: arrivo classe 1
        clock = nextArrival1;

        % Accetto sempre classe 1 (scelta coerente con la regola di protezione)
        count1 = count1 + 1;

        % Pianifico il prossimo arrivo classe 1
        nextArrival1 = clock + exprnd(1/trueArrivalRate1);

    else
        % Prossimo evento: arrivo classe 2
        clock = nextArrival2;

        % Accetto classe 2 solo se non supero il booking limit
        % In pratica: lascio almeno protectionLevel posti "liberi" per la classe 1
        if count2 < cap - protectionLevel
            count2 = count2 + 1;
        end

        % Pianifico il prossimo arrivo classe 2
        nextArrival2 = clock + exprnd(1/trueArrivalRate2);
    end

    % Controllo capacita'
    if count1 + count2 >= cap
        full = true;
    end

    % Aggiorno stop: la partenza avviene al tempo T
    stop = min(nextArrival1, nextArrival2) > T;
end

% Profitto totale realizzato
fprintf(1,'Total profit %d\n', count1*price1 + count2*price2);

% POSSIBILI MIGLIORAMENTI
% 1) Una volta raggiunto il booking limit per la classe 2, potrei ignorare i suoi arrivi
%    e campionare direttamente il numero di arrivi classe 1 con una Poisson.
% 2) Incapsulare in una funzione (con rate generici) per fare molte repliche e stimare media/
    varianza del profitto.
% 3) Confrontare i risultati anche con un modello statico "ideale" (senza simulazione).
% 4) La scelta del quantile Poisson puo' essere discutibile: quale quantile e' piu' corretto e
    perche'?
```

Conclusioni Lo script implementa una regola di controllo capacità basata su un **livello di protezione** per la classe alta e stima il profitto realizzato tramite simulazione a eventi discreti.

(i) Protezione e trade-off ricavo. La protezione riduce il rischio di "sprecare" capacità vendendola troppo presto alla classe economica, ma può anche generare rifiuti e posti vuoti se la domanda di classe 1 non si materializza.

(ii) Errore di stima. Il livello di protezione viene deciso con tassi stimati: se $\hat{\lambda}_1$ è troppo alto si

rischia di rifiutare troppa classe 2; se è troppo basso si rischia di saturare con classe 2 e poi non avere spazio per classe 1.

(iii) Struttura del simulatore. Il modello è un simulatore a eventi con due processi di arrivo indipendenti; termina quando l'aereo è pieno oppure quando si supera l'orizzonte T.

In sintesi, il codice è un primo prototipo utile per studiare l'effetto del booking limit sul profitto, ma per un'analisi affidabile servono più repliche, confronto con benchmark (statico/dinamico) e una riflessione sulla scelta del quantile usato per la protezione.

12.7 Roll queue: simulazione a eventi discreti con buffer e domanda batch (code 7)

Contesto del problema Lo script simula un sistema di tipo **queueing/inventory** in cui si preparano "roll" (panini) uno alla volta e i clienti arrivano nel tempo richiedendo un numero intero di roll.

La produzione dei roll avviene con tempi di preparazione **uniformi** tra `lowTime` e `highTime`. I roll pronti vengono accumulati in un **buffer** con capacità massima `maxPlaces`: quando il buffer è pieno, la produzione viene **bloccata** (non si preparano ulteriori roll finché non si libera spazio).

I clienti arrivano con tempi tra arrivi **esponenziali** (media `meanInterTime`) e ogni cliente richiede una quantità casuale di roll (`demand`) distribuita **uniformemente discreta** tra 1 e `maxDemand`.

Se al momento dell'arrivo c'è già una coda (clienti in attesa) si entra in coda; se non c'è coda, il cliente prova a soddisfare immediatamente la domanda con il buffer disponibile. La performance misurata è il **tempo medio di attesa** dei clienti serviti.

```
%% 7 - Roll queue (Es. 3 tema 03/2024)
% Unita' di tempo: 1 minuto
% Idea: i roll vengono preparati (uno alla volta) e accumulati in un buffer finito.
% I clienti arrivano e richiedono una quantita' intera di roll (domanda batch).
% Se il buffer e' pieno, la produzione si blocca finche' non si libera spazio.

% Parametri di preparazione roll (Uniforme)
lowTime  = 1.5;
highTime = 2;

% Buffer
maxPlaces = 6;    % capacita' massima del buffer (roll pronti)

% Arrivi clienti e domanda
meanInterTime = 4;  % media dei tempi tra arrivi (Esponenziale)
maxDemand = 3;      % domanda massima per cliente (Uniforme discreta 1..maxDemand)

% Contatori
count   = 0;      % clienti serviti
toServe = 1000;   % numero di clienti da servire

% Stato del sistema
clock = 0;

% Prossimi eventi (due processi indipendenti)
nextArrival = exprnd(meanInterTime);         % prossimo arrivo cliente
nextRoll    = unifrnd(lowTime,highTime);     % prossimo completamento roll

buffer  = 3;       % stato iniziale: roll gia' pronti
blocked = false;   % se true: produzione bloccata per buffer pieno

% Coda clienti (traccio: tempo di ingresso e domanda residua)
joinTime       = [];   % istante di ingresso in coda di ciascun cliente
```

```
residualDemand = [];   % domanda residua per ciascun cliente in coda

% Statistica: tempo totale di attesa (solo per clienti che effettivamente attendono)
totalWaitingTime = 0;

while count <= toServe

    % Evento 1: completamento di un roll (se avviene prima del prossimo arrivo)
    if nextRoll <= nextArrival
        clock = nextRoll;

        if length(residualDemand) > 0
            % C'e' almeno un cliente in coda: uso il roll appena pronto per servire il primo
            residualDemand(1) = residualDemand(1) - 1;

            if residualDemand(1) == 0
                % Cliente completato: aggiorno statistiche e rimuovo dalla coda
                totalWaitingTime = totalWaitingTime + (clock - joinTime(1));
                count = count + 1;

                joinTime(1) = [];
                residualDemand(1) = [];
            end

        else
            % Nessun cliente in attesa: metto il roll nel buffer
            buffer = buffer + 1;
        end

        % Gestione del blocco: se buffer pieno, fermo la produzione
        if buffer == maxPlaces
            blocked  = true;
            nextRoll = inf;   % nessun altro completamento finche' non si sblocca
        else
            nextRoll = clock + unifrnd(lowTime,highTime);
        end

    % Evento 2: arrivo di un cliente
    else
        clock = nextArrival;
        nextArrival = clock + exprnd(meanInterTime);

        % Domanda del cliente (Uniforme discreta)
        demand = unidrnd(maxDemand);

        if length(residualDemand) > 0
            % Se c'e' gia' coda, il cliente entra in fondo
            residualDemand(end+1) = demand;
            joinTime(end+1) = clock;

        else
            % Se non c'e' coda, provo a servire subito con il buffer
            if buffer >= demand
                buffer = buffer - demand;
                count = count + 1;  % attesa nulla
```

```
            else
                % Buffer insufficiente: consumo tutto il buffer e il resto diventa attesa
                demand = demand - buffer;
                buffer = 0;

                residualDemand(1) = demand;
                joinTime(1) = clock;
            end

            % Se la produzione era bloccata e ho liberato spazio, la riattivo
            if blocked
                blocked  = false;
                nextRoll = clock + unifrnd(lowTime,highTime);
            end
        end
    end
end

fprintf(1,'average waiting time = %.2f\n', totalWaitingTime/count);
```

Conclusioni Lo script costruisce un simulatore a **eventi discreti** con due processi concorrenti (arrivi clienti e completamenti di roll) e un **buffer finito** che può bloccare la produzione.

(i) Interazione tra coda e buffer. Se non ci sono clienti in attesa, i roll vengono accumulati; se arrivano clienti con domanda batch, il buffer può azzerarsi rapidamente e generare attesa (domanda residua).

(ii) Meccanismo di blocco. Quando il buffer è pieno, la produzione viene fermata mettendo `nextRoll = inf`; la produzione riparte solo quando un cliente consuma roll e libera spazio (`blocked` torna `false`).

(iii) Misura di performance. Il tempo medio di attesa viene stimato tramite la somma dei tempi in coda (`clock - joinTime`) per i clienti che attendono. I clienti serviti immediatamente contribuiscono con attesa nulla.

In sintesi, il modello è utile per studiare l'effetto combinato di **capacità del buffer**, **variabilità di produzione** e **domanda batch** sul congestionamento e sui tempi di attesa medi.

12.8 Cutting stock: formulazione simmetrica e generazione di colonne (code 8a, 8b, 8c)

Contesto del problema Il **cutting stock problem** consiste nel tagliare rotoli (o tronchi) di lunghezza fissa in pezzi di lunghezze predefinite per soddisfare una domanda assegnata, minimizzando il numero di rotoli utilizzati (o, in modo equivalente, lo spreco totale).

Il problema è noto per la sua elevata complessità combinatoria e per la presenza di numerose simmetrie, che rendono rapidamente impraticabili formulazioni dirette quando la dimensione dell'istanza cresce.

Vengono confrontati tre approcci: **(8a)** una formulazione compatta e simmetrica di tipo Kantorovich, **(8b)** una risoluzione tramite **column generation** con approccio *problem-based*, e **(8c)** la stessa tecnica di column generation implementata in versione *solver-based*. Le versioni (8b) e (8c) risolvono lo **medesimo modello matematico**, ma differiscono per livello di astrazione e controllo sul solver.

```
%% 8a - Cutting stock: formulazione simmetrica (Kantorovich) + symmetry breaking

% Generazione casuale di un'istanza
rng 'default'
rollLength = 1000;
numLenghts = 8;
lenghts = randsample(10:800, numLenghts);
```

```
demand  = 20 + unidrnd(150, numLenghts, 1);

% Stima pessimistica del numero massimo di rotoli
fudgeFactor = 2;
numRolls = ceil(fudgeFactor*(dot(lenghts,demand)/rollLength));

% Formulazione simmetrica
cutStock = optimproblem("ObjectiveSense","minimize");

% delta(j)=1 se il rotolo j viene usato
delta  = optimvar("delta", numRolls, 1, "Type","integer", ...
                  "LowerBound",0, "UpperBound",1);

% numCut(i,j)=pezzi di tipo i tagliati dal rotolo j
numCut = optimvar("numCut", numLenghts, numRolls, ...
                  "Type","integer", "LowerBound",0);

cutStock.Objective = sum(delta,'all');
cutStock.Constraints.meetDemand = sum(numCut,2) >= demand;
cutStock.Constraints.useLog = lenghts*numCut <= rollLength*delta';

fprintf(1,"Running symmetric model formulation\n")
solve(cutStock);

% Vincoli di symmetry breaking: uso ordinato dei rotoli
breaking = optimconstr(numRolls-1,1);
for k = 1:(numRolls-1)
    breaking(k) = delta(k+1) <= delta(k);
end
cutStock.Constraints.breaking = breaking;

fprintf(1,"Running symmetry breaking model formulation\n")
solve(cutStock);
```

Osservazioni su (8a) La formulazione simmetrica è intuitiva e aderente alla descrizione del problema, ma introduce un numero molto elevato di variabili e soffre di **simmetria**: rotoli indistinguibili generano molte soluzioni equivalenti. I vincoli di symmetry breaking migliorano le prestazioni, ma la scalabilità rimane limitata, come evidenziato dai tempi di risoluzione e dal numero di nodi esplorati.

```
%% 8b - Cutting stock: column generation (problem-based)

logLength  = 40;
lengthlist = [8; 12; 16; 20];
quantity   = [90; 111; 55; 30];
nLengths   = length(lengthlist);

% Pattern iniziali (uno per ciascuna lunghezza)
patterns  = diag(floor(logLength ./ lengthlist));
nPatterns = size(patterns,2);

% Sottoproblema di generazione pattern
subproblem = optimproblem();
cuts = optimvar('cuts', nLengths, 1, 'Type','integer', ...
                'LowerBound', zeros(nLengths,1));
subproblem.Constraints = dot(lengthlist,cuts) <= logLength;
```

```
lpopts = optimoptions('linprog','Display','off');
ipopts = optimoptions('intlinprog',lpopts);

reducedCost = -inf;
reducedCostTolerance = -0.0001;
exitflag = 1;

while reducedCost < reducedCostTolerance && exitflag > 0
    logprob = optimproblem('Description','Cut Logs');
    x = optimvar('x', nPatterns, 1, 'LowerBound', 0);

    logprob.Objective.logsUsed = sum(x);
    logprob.Constraints.Demand = patterns*x >= quantity;

    [values,nLogs,exitflag,~,lambda] = solve(logprob,'options',lpopts);

    if exitflag > 0
        fprintf('Using %g logs\n',nLogs);
        subproblem.Objective = 1 - dot(lambda.Constraints.Demand,cuts);
        [values,reducedCost,pexitflag] = solve(subproblem,'options',ipopts);
        newpattern = round(values.cuts);

        if pexitflag > 0 && reducedCost < reducedCostTolerance
            patterns = [patterns newpattern];
            nPatterns = nPatterns + 1;
        end
    end
end

x.Type = 'integer';
[values,logsUsed] = solve(logprob,'options',ipopts);
values.x = round(values.x);
fprintf('Optimal solution uses %g logs\n',sum(values.x));
```

```
%% 8c - Cutting stock: column generation (solver-based)

% Dati del problema: tronchi/rotoli tutti della stessa lunghezza
logLength  = 40;
lengthlist = [8; 12; 16; 20];     % lunghezze dei pezzi richiesti
quantity   = [90; 111; 55; 30];   % domanda per ciascuna lunghezza
nLengths   = length(lengthlist);

% Pattern iniziali "banali": per ogni lunghezza taglio il massimo numero possibile
% (matrice diagonale: ogni colonna = un pattern diverso)
patterns  = diag(floor(logLength ./ lengthlist));
nPatterns = size(patterns,2);

% Sottoproblema (pricing): genera un nuovo pattern risolvendo un ILP
% variabili = numero di pezzi di ciascuna lunghezza nel nuovo pattern
lb2 = zeros(nLengths,1);
A2  = lengthlist';  % vincolo di capacità: somma(length_i * cuts_i) <= logLength
b2  = logLength;

% Opzioni solver (silenzia output)
```

```
lpopts = optimoptions('linprog','Display','off');
ipopts = optimoptions('intlinprog',lpopts);

% Inizializzazione criterio di arresto della column generation
reducedCost = -Inf;
reducedCostTolerance = -0.0001; % aggiungo una colonna solo se costo ridotto < 0 (con
    tolleranza)
exitflag = 1;

% Column generation: alterno
% (1) master LP (con pattern correnti) e (2) pricing ILP (nuovo pattern)
while reducedCost < reducedCostTolerance && exitflag > 0

    % Master LP: min sum(x) s.t. patterns*x >= quantity, x >= 0
    lb = zeros(nPatterns,1);
    f  = lb + 1;          % costo unitario di ogni pattern: 1 (minimizza numero di tronchi)
    A  = -patterns;       % trasformo >= in <= per linprog
    b  = -quantity;

    [values,nLogs,exitflag,~,lambda] = linprog(f,A,b,[],[],lb,[],lpopts);

    if exitflag > 0
        fprintf('Using %g logs\n',nLogs);

        % Pricing problem: cerco un pattern con costo ridotto negativo
        % Nel solver-based uso direttamente i moltiplicatori duali del master LP:
        % f2 = -lambda, così il sottoproblema cerca di minimizzare il costo ridotto
        f2 = -lambda.ineqlin;

        [values,reducedCost,pexitflag] = intlinprog(f2,1:nLengths,A2,b2,[],[],lb2,[],ipopts);

        % Il costo ridotto effettivo è: 1 + reducedCost (perché nel master ogni colonna ha
    costo 1)
        reducedCost = 1 + reducedCost;
        newpattern = round(values); % pattern trovato (intero)

        % Se esiste un pattern migliorativo (costo ridotto < 0), lo aggiungo come nuova colonna
        if pexitflag > 0 && reducedCost < reducedCostTolerance
            patterns  = [patterns newpattern];
            nPatterns = nPatterns + 1;
        end
    end
end
```

Conclusioni Il blocco (8a–8c) evidenzia tre livelli di modellazione dello stesso problema.

(i) Formulazione simmetrica (8a). È concettualmente semplice e utile per comprendere il problema, ma soffre di forti simmetrie e di scarsa scalabilità.

(ii) Column generation (8b–8c). Evita l'enumerazione esplicita dei pattern e costruisce la soluzione in modo incrementale tramite un master LP e un sottoproblema ILP, seguendo un criterio di arresto analogo a quello del metodo del simplesso.

(iii) Problem-based vs solver-based. Le versioni (8b) e (8c) implementano lo stesso algoritmo di column generation: la prima privilegia chiarezza e modularità, la seconda offre maggiore controllo sulle strutture matriciali e sul solver. I risultati coincidono, confermando l'equivalenza matematica delle due implementazioni.

In sintesi, (8a) è utile a fini didattici, mentre (8b–8c) rappresentano l'approccio standard ed efficiente per istanze realistiche di cutting stock.