Business Analytics

1 CAP 1 – Introduzione alle decisioni in condizioni di incertezza

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. analisi di fenomeni osservati come l'overbooking nel trasporto aereo).
- **Business Analytics predittiva**: mira a stimare eventi o variabili future sulla base di modelli statistici o probabilistici, utilizzando informazioni storiche e ipotesi sul comportamento del sistema (e.g. distinzione tra *forecasting* e *prediction*).
- **Business Analytics prescrittiva**: si concentra sul supporto alle decisioni, indicando quale azione dovrebbe essere intrapresa per ottimizzare una misura di prestazione in presenza di vincoli e incertezza (e.g. problemi di *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.

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 $L(x,X) = c_u(X-x)^+ + c_o(x-X)^+$, dove $(\cdot)^+ = \max\{0,\cdot\}$, mentre c_u e c_o rappresentano rispettivamente il **costo di sottostima e di sovrastima**. Il problema decisionale associato è

$$\min_{x\in\mathbb{R}} \ \mathbb{E}\big[c_u(X-x)^+ + c_o(x-X)^+\big].$$

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

$$\boxed{\min_{x \in X} \max_{\xi \in U} f(x, \xi)}$$

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

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

Questa formulazione mette in evidenza che, in generale, $\mathbb{E}\big[f(x, \tilde{\xi})\big] \neq f\big(x, \mathbb{E}[\tilde{\xi}]\big)$, e giustifica la necessità di modelli decisionali che tengano esplicitamente conto dell'incertezza.

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 $q = \mathbb{E}\big[D\big]$. 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**).

Consideriamo una **formulazione analitica** del problema, dove assumiamo che la domanda abbia distribuzione nel continuo con densità $f_D(x)$ nota. Il valore atteso del profitto è:

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

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

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

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

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

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

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

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

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

La **condizione di ottimalità** prescrive un valore di q^* tale da ottenere una data probabilità di non andare in *stockout*, che è una misura del **livello di servizio**. Per valori di m molto grandi rispetto a c_u, il livello di servizio ottimale tende al 100%, e quindi si acquista/produce molta merce, mentre esso si riduce se il margine è basso.

Esempio: newsvendor nel caso normale Se assumiamo domanda normale, con valore atteso μ e deviazione standard σ, le note proprietà dei quantili di una distribuzione normale ci permettono di scrivere $q^* = \mu + z_\beta \sigma$, dove z_β è il quantile della normale standard corrispondente al livello di probabilità

$$\beta = \frac{m}{m + c_u}.$$

È facile vedere che il solo caso in cui è ottimale ordinare il valore atteso μ della domanda si verifica per penalità simmetriche, $m = c_u$. Si potrebbe pensare che all'aumentare dell'incertezza della domanda si debba essere conservativi e ordinare meno del valore atteso. In realtà non è così in generale.

Esiste anche un altro modo, del tutto equivalente, di ricavare la regola di decisione, introducendo una **funzione di perdita**

$$L(q, D) = m(D - q)^+ + c_u(q - D)^+,$$

legata alla discrepanza tra q e D, dove $(x)^+ \doteq \max\{x, 0\}$.

Si può dimostrare che il profitto atteso e il valore atteso della perdita differiscono per una costante:

$$\mathbb{E}[\pi(q, D)] = m\mathbb{E}[D] - \mathbb{E}[L(q, D)].$$

Massimizzare il profitto atteso è quindi equivalente a minimizzare il valore atteso della funzione di perdita, che rappresenta una penalità legata all'impatto economico della discrepanza tra q e D. Se si scegliesse q come previsione della domanda, si dovrebbe scegliere una funzione di perdita opportuna:

- nel caso di una perdita quadratica, $L(q, D) = (q - D)^2$, è facile dimostrare che la soluzione ottima sarebbe $q^* = \mathbb{E}[D]$;
- nel caso di una perdita data dalla deviazione assoluta, $L(q, D) = |q - D|$, usando la regola di Leibniz è facile dimostrare che la soluzione ottima è data dalla **mediana della domanda**, ovvero il quantile al livello di probabilità 50%.

Nel caso del newsvendor, la penalità è lineare a tratti, come la funzione valore assoluto, ma le due pendenze non sono simmetriche.

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

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

dove $(x)^+ \doteq \max\{x, 0\}$. 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 $c_o = c - r$.

Se l'incertezza sulla domanda è modellata da una variabile casuale continua con densità $f_D(x)$, il profitto atteso è

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

Applicando nuovamente la regola di Leibniz, la condizione di ottimalità del primo ordine porta a

$$c_u\big(1 - F_D(q)\big) - c_o F_D(q) = 0,$$

da cui segue che la quantità ottima q^* soddisfa

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

Questa condizione è facile da interpretare: la quantità ottima da ordinare è un **quantile della distribuzione della domanda**, che dipende dall'economia del problema. Quando c_u è grande rispetto a c_o, la quantità ottima è elevata; quando i due coefficienti di penalità sono uguali, la soluzione ottima coincide con la mediana della distribuzione della domanda.

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

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

e un vincolo probabilistico congiunto

$$\mathbb{P}\{g(x, \tilde{\xi}) \leq 0_m\} \geq 1 - \alpha,$$

dove 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, come mostrato nelle slide tramite un esempio di vincoli di capacità, 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 Proprietà di convessità dei modelli di programmazione stocastica

2 CAP 2 – Elementi di complessità computazionale

Recap La **complessità computazionale** dipende dalla dimensione dell'istanza e può crescere in modo fattoriale, esponenziale o polinomiale. Piccole variazioni nella struttura di un problema possono modificare radicalmente la sua complessità, come nel caso dei problemi di scheduling $1//L_{\max}$ e $1/r_i/L_{\max}$.

La **teoria della NP-completezza** introduce le classi P, NP, NPH e NPC e il concetto di riduzione polinomiale per confrontare la difficoltà dei problemi. I problemi NP-completi sono equivalenti in termini di complessità e l'esistenza di un algoritmo polinomiale per uno di essi implicherebbe $P = NP$.

La riduzione da subset-sum mostra che la versione decisionale di $1/r_i/L_{\max}$ è NP-completa e che il corrispondente problema di ottimizzazione è NP-difficile. **La codifica dei dati è cruciale**: l'algoritmo per il knapsack ha complessità pseudo-polinomiale, poiché dipende dal valore numerico dei dati e non dalla loro lunghezza binaria.

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. Supponiamo che esista una soluzione ottima in cui, per due job consecutivi in sequenza, prima $i = \sigma(k)$ e poi $j = \sigma(k+1)$, si abbia $d_i > d_j$. Indichiamo con C_i e $C_j = C_i + p_j$ i due tempi

di completamento nella soluzione corrente, per la quale abbiamo due valori di lateness $L_i = C_i - d_i$ e $L_j = C_j - d_j$.
Se scambiamo i due job, avremo $C'_j < C_j$ e $C'_i = C_j$. Per il job j, che anticipiamo, abbiamo $C'_j < C_j$ e $L'_j < L_j$, e quindi la lateness del job j non può che migliorare nella nuova soluzione. Per il job i, che viene spostato in avanti nella sequenza, abbiamo

$$L'_i = C'_i - d_i = C_j - d_i < C_j - d_j = L_j,$$

quindi il job i peggiora la sua lateness, che però non potrà essere peggio della vecchia lateness del job j. La nuova soluzione migliora quella precedente, contraddicendo l'assunzione di ottimalità.

Abbiamo quindi un algoritmo di complessità polinomiale per il problema. **Ma cosa accade se complichiamo leggermente il problema, introducendo dei tempi di rilascio (release time o ready time)** r_i**,** $i \in [n]$**, dei job?** Per il problema $1/r_i/L_{\max}$ non sono noti algoritmi di complessità polinomiale, ed è facile costruire controesempi alla regola EDD (da adattare comunque alla disponibilità di job).
Esistono algoritmi **branch-and-bound** per il problema $1/r_i/L_{\max}$ (quindi complessità esponenziale). Non si conosce un algoritmo di complessità polinomiale per questo problema di ottimizzazione combinatoria (e per molti altri), ma neppure è stato dimostrato che esso non possa esistere.

2.1 Caratterizzazione della complessità di problemi: classi P, NPC e NPH

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

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 $\sum_{i \in J} a_i = b$?

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 $\min_{x \in S} f(x)$, **possiamo definirne una versione decisionale**, scegliendo un numero k e chiedendoci se esiste $x \in S$ tale che $f(x) \leq k$, dove k è un numero intero. Indichiamo con PO il problema di ottimizzazione, e con PD il corrispondente problema di decisione.
Se abbiamo a disposizione un algoritmo efficiente per PO, allora possiamo risolvere in modo efficiente anche PD: basta risolvere il problema di ottimizzazione e verificare se $f(x^*) \leq k$. Questo ci 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 possiamo 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 abbiamo un upper bound U sul costo ottimo, possiamo applicare una procedura di bisezione.
Indichiamo 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 vogliamo 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 enumeriamo, mediante un albero di ricerca, tutti i possibili sottoinsiemi J, possiamo verificare se una specifica istanza ha risposta positiva o negativa, ma chiaramente tale algoritmo ha **complessità esponenziale**.
Tuttavia, se disponessimo di un ipotetico **calcolatore non deterministico**, in grado di eseguire un numero infinito di processi di calcolo in parallelo, saremmo 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. Diremo che P è riducibile in tempo polinomiale a Q, e useremo la notazione $P \prec Q$.) 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, potremmo 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. Indichiamo con NPH la classe dei problemi NP-difficili.

Problemi NP-completi Un problema di decisione P è detto NP-completo se è in Np ed è NP-hard. Indichiamo con NNPC la classe dei problemi NP-completi. Le implicazioni pratiche 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;
- Che un problema NP-completo Q può essere ridotto in tempo polinomiale a P.

Osserviamo che, dato che Q è NP-completo, $P \prec Q$, e se trascuriamo 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 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. Avremmo 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 accettiamo l'ipotesi $P \neq NP$, possiamo 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.

Se dimostriamo che un problema P è in NPC, questo può a sua volta essere trasformato in altri problemi, permettendoci 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 ci fornisce la soluzione.

Teorema di Cook. 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.
Il fatto che tale problema faccia parte della classe NPC ci permette di dimostrare il teorema seguente, che risolve la questione da cui siamo partiti.

Teorema. Consideriamo 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, $j \in [n]$, creiamo n job J_j con parametri

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

Creiamo 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

Il teorema dimostra che un problema di decisione legato al problema di ottimizzazione $1/r_i/L_{\max}$ è NP-completo, **ma cosa possiamo dire del problema di ottimizzazione stesso?**
Per definizione, la classe NPC contiene solo problemi di decisione. Possiamo però 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 possiamo includere problemi di ottimizzazione ai quali possiamo 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 abbiamo assunto che i dati fossero numeri interi. Ma il problema a numeri interi può evidentemente essere ridotto al problema generale. Possiamo quindi affermare che il problema di scheduling $1/r_i/L_{\max}$ è NP-difficile.

L'impatto della codifica di un problema Nella trattazione ci siamo limitati alle classi fondamentali e siamo stati piuttosto informali e imprecisi. Non possiamo però 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, **consideriamo un classico algoritmo di programmazione dinamica** per la soluzione del problema knapsack:

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

Definiamo la funzione valore

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

quando la capacità residua è s. In sostanza, la funzione valore assume che siano già state fatte scelte di inserimento o meno degli oggetti da 1 a $k-1$; a valle di tale selezione, abbiamo una capacità residua s, e ci chiediamo 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 che abbiamo considerato avrebbe **complessità polinomiale**. Si dice infatti che un algoritmo di questo tipo è *pseudo-polinomiale*.

3 CAP 3 –Metodi di decomposizione in ottimizzazione

3.1 Motivazione

I metodi di decomposizione giocano un ruolo prominente nell'ottimizzazione, in quanto consentono di:

- **Sfruttare una struttura favorevole**. Ad esempio, problemi con sottoproblemi di rete (*network sub problems*) che possono essere risolti in tempo polinomiale con algoritmi specifici e particolarmente efficienti.
- **Parallelizzare** la soluzione di problemi su larga scala.
- Affrontare modelli di ottimizzazione stocastica su larga scala e basati su scenari.
- Affrontare problemi combinatori complessi gestendo una sequenza di sottoproblemi più semplici, eventualmente mescolando diverse strategie di soluzione (decomposizione sequenziale).
- **Evitare problemi di modellazione** legati a vincoli molto difficili.

3.1.1 Strutture Complicanti nei Modelli

Un problema di ottimizzazione generale:

$$\text{opt} \quad f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{x} \in \mathcal{S}$$

Se il problema è della forma:

$$\text{opt} \quad \sum_{j \in [n]} f_j(\mathbf{x}_j) \quad \text{s.t.} \quad \mathbf{x}_j \in \mathcal{S}_j, \quad j \in [n]$$

allora può essere facilmente decomposto in sottoproblemi disaccoppiati $\text{opt}_{\mathbf{x}_j \in \mathcal{S}_j} f_j(\mathbf{x}_j)$. Tuttavia, spesso è presente un **fattore complicante**.

Struttura a Blocchi Angolare Un Modello di Programmazione Lineare (LP) su larga scala può presentare una struttura a blocchi angolare nella matrice tecnologica A.

- **Vincoli di Interazione Complicanti**: Nel primo caso (struttura a blocchi angolare), un insieme di vincoli di interazione accoppia i sottoproblemi e impedisce la decomposizione. Si può ricorrere alla **decomposizione duale lagrangiana**.
- **Variabili di Interazione Complicanti**: Nel secondo caso (struttura a blocchi a doppia angolazione), abbiamo variabili di interazione che impediscono la decomposizione. Si può decomporre il problema se si fissano le variabili di interazione (es. decomposizione a forma di L, che è la **Decomposizione di Benders** per la programmazione stocastica).

3.2 Metodi Interconnessi di Decomposizione

Esistono diversi metodi di decomposizione interconnessi, alcuni esatti, altri approssimati:

- Rilassamento Lagrangiano e Decomposizione Lagrangiana.
- Euristiche Duali.
- **Decomposizione di Dantzig-Wolfe**.
- Generazione di Colonne (*Column Generation*).
- Decomposizione Gerarchica.
- MatHeuristics (non da confondere con MetaHeuristics).
- **Decomposizione di Benders** per MILP con struttura speciale.
- Decomposizione a forma di L (*L-shaped decomposition*) per programmazione stocastica a due stadi con ricorso.
- *Progressive Hedging* per programmazione stocastica multi-stadio.
- Programmazione Dinamica (decomposizione basata sul tempo).

3.3 Decomposizione Duale (Lagrangiana)

La decomposizione duale è utile quando il fattore complicante è un vincolo di interazione.

Problema Primale (P) Si consideri un problema del tipo:

$$\max \sum_{i=1}^{n} f_i(\mathbf{x}_i) \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{n} g_i(\mathbf{x}_i) \leq b \tag{2}$$ (Vincolo di Bilancio/Interazione)

$$\mathbf{x}_i \in \mathcal{S}_i, \quad i = 1, \ldots, n \tag{3}$$ (Vincoli Locali Disaccoppiati)

Il vincolo (2) accoppia le decisioni $\mathbf{x}_i$ e impedisce una facile risoluzione.

Funzione Lagrangiana Dualizzando il vincolo di bilancio (2) con un moltiplicatore $\mu \geq 0$, si ottiene la funzione Lagrangiana:

$$\mathcal{L}(\mathbf{x}, \mu) = \sum_{i=1}^{n} f_i(\mathbf{x}_i) + \mu \left(b - \sum_{i=1}^{n} g_i(\mathbf{x}_i) \right) = \sum_{i=1}^{n} \left[f_i(\mathbf{x}_i) - \mu g_i(\mathbf{x}_i) \right] + \mu b$$

Sottoproblemi Disaccoppiati Per un μ fissato (prezzo ombra della risorsa), il problema si scompone in n sottoproblemi indipendenti:

$$\max_{\mathbf{x}_i \in \mathcal{S}_i} \left[f_i(\mathbf{x}_i) - \mu g_i(\mathbf{x}_i) \right], \quad i = 1, \ldots, n$$

Ogni sottoproblema massimizza il contributo di profitto meno il costo della risorsa (valutato a μ).

Problema Duale e Coordinamento La Funzione Duale è $W(\mu) = \max_{\mathbf{x} \in \mathcal{S}} \mathcal{L}(\mathbf{x}, \mu)$. Poiché il problema primale (P) era di massimo, il Problema Duale (D) cerca un *lower bound* sull'ottimo primale ed è espresso come:

$$\min_{\mu \geq 0} W(\mu) \qquad \text{(Problema Coordinatore)}$$

Il coordinatore regola il prezzo μ per portare i sottodecisori locali a soddisfare il vincolo di interazione.

Sottogradiente Dato che la funzione duale $W(\mu)$ è convessa, essa è sempre non differenziabile. Il sottogradiente è dato da:

$$\sum_{i=1}^{n} g_i(\mathbf{x}_i^*) - b$$

dove $\mathbf{x}_i^*$ è la soluzione dei sottoproblemi.

- Se il sottogradiente è positivo (il budget è superato), si deve **aumentare** il prezzo della risorsa (μ).
- Se il sottogradiente è negativo (il budget non è superato), si deve **diminuire** il prezzo della risorsa (μ).

3.4 Decomposizione a Forma di L e Tagli di Ottimalità/Fattibilità

La Decomposizione a forma di L è l'applicazione del metodo di **Decomposizione di Benders** alla programmazione stocastica a due stadi con ricorso.

3.4.1 Programmazione Stocastica a Due Stadi (SLP)

Si consideri il problema SLP con ricorso:

$$\min \quad \mathbf{c}^T\mathbf{x} + \mathbb{E}_\xi[Q(\mathbf{x},\xi)] \quad \text{s.t.} \quad A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq 0$$

dove $Q(\mathbf{x},\xi)$ è la funzione di ricorso per uno scenario ξ:

$$Q(\mathbf{x},\xi) \equiv \min_{\mathbf{y}} \quad \mathbf{q}(\xi)^T\mathbf{y} \quad \text{s.t.} \quad W(\xi)\mathbf{y} = \mathbf{h}(\xi) - T(\xi)\mathbf{x}, \quad \mathbf{y} \geq 0$$

La funzione di ricorso (o costo futuro atteso) $Q(\mathbf{x}) \equiv \mathbb{E}_\xi[Q(\mathbf{x},\xi)]$ è **convessa** per distribuzioni di probabilità discrete ed è poliedrica.

3.4.2 Decomposizione a Forma di L (Benders)

Il problema equivalente deterministico (che minimizza il costo totale) può essere riscritto nel **Problema Master Rilassato (RMP)**:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} + \theta \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \quad \mathbf{x} \geq 0 \\ & \theta \geq \text{Tagli di Ottimalità e Fattibilità} \end{aligned}$$

dove θ è un'approssimazione dal basso della funzione di ricorso $Q(\mathbf{x})$.

Tagli di Ottimalità I tagli di ottimalità approssimano la funzione di ricorso convessa $Q(\mathbf{x})$ utilizzando un'iperpiano di supporto (concetto di *cutting plane* di Kelley). Dato un $\hat{\mathbf{x}}$ ottimale dal RMP, si risolve il duale del sottoproblema del secondo stadio per ogni scenario s:

$$Q_s(\hat{\mathbf{x}}) \equiv \max_{\pi_s} \quad (\mathbf{h}_s - T_s\hat{\mathbf{x}})^T\pi_s \quad \text{s.t.} \quad W^T\pi_s \leq \mathbf{q}_s$$

Sia $\hat{\pi}_s$ la soluzione duale ottima. Il **Taglio di Ottimalità** (o Taglio di Benders) è:

$$\theta \geq \sum_{s\in\mathcal{S}} p_s(\mathbf{h}_s - T_s\mathbf{x})^T\hat{\pi}_s$$

Questo taglio viene aggiunto al RMP per migliorare l'approssimazione di $\theta \geq Q(\mathbf{x})$.

Tagli di Fattibilità Se la fase di ricorso non è completa, per una data decisione $\hat{\mathbf{x}}$ alcuni sottoproblemi potrebbero essere infattibili. In questo caso, il duale del sottoproblema per lo scenario infattibile è illimitato. Si cerca un raggio estremo (extreme ray) π^* del set ammissibile duale $W^T\pi_s \leq \mathbf{q}_s$. Il **Taglio di Fattibilità** (o Taglio di Benders) aggiunto al RMP è:

$$(\pi^*)^T(\mathbf{h}_s - T_s\mathbf{x}) \leq 0$$

Questo vincolo elimina $\mathbf{x}$ dal RMP, garantendo che il sottoproblema sia fattibile.

3.5 Decomposizione di Dantzig-Wolfe (D-W)

La decomposizione di D-W e la **Generazione di Colonne** sono metodi che si applicano quando il fattore complicante è un insieme di vincoli di interazione (vincoli "cattivi").

Problema Primale (P) Si consideri un modello LP con vincoli divisi in "cattivi" (riga 12) e "facili" (riga 13):

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq \mathbf{b} && (12) \quad \text{(Vincoli Complicanti)} \\ & D\mathbf{x} \geq \mathbf{f} && (13) \quad \text{(Vincoli Facili)} \\ & \mathbf{x} \geq 0 \end{aligned}$$

Sia $X = \{\mathbf{x} \in \mathbb{R}^n : D\mathbf{x} \geq \mathbf{f}, \mathbf{x} \geq 0\}$ la regione ammissibile dei vincoli facili.

Rappresentazione Poliedrica Poiché X è un poliedro, qualsiasi vettore $\mathbf{x} \in X$ può essere espresso come una combinazione convessa dei suoi punti estremi $\mathbf{v}_q \in V$ e una combinazione conica delle sue direzioni estreme $\mathbf{d}_r \in D'$:

$$\mathbf{x} = \sum_{q \in V} \lambda_q \mathbf{v}_q + \sum_{r \in D'} \mu_r \mathbf{d}_r$$

con i vincoli di convessità e non-negatività:

$$\sum_{q \in V} \lambda_q = 1; \quad \lambda_q \geq 0, \mu_r \geq 0$$

Problema Master Ristretto (RMP) Sostituendo $\mathbf{x}$ nel problema primale (P) si ottiene un problema di ottimizzazione nelle nuove variabili λ_q e μ_r, noto come Problema Master. Poiché il numero di punti e direzioni estreme può essere enorme, si inizia con un sottoinsieme limitato $V_0 \subset V$ e $D'_0 \subset D'$ e si risolve il **Problema Master Ristretto (RMP)**:

$$\begin{aligned}
\min \quad & \sum_{q \in V_0} \mathbf{c}^T \mathbf{v}_q \lambda_q + \sum_{r \in D'_0} \mathbf{c}^T \mathbf{d}_r \mu_r \\
\text{s.t.} \quad & \sum_{q \in V_0} A\mathbf{v}_q \lambda_q + \sum_{r \in D'_0} A\mathbf{d}_r \mu_r \geq \mathbf{b} && (\pi) \\
& \sum_{q \in V_0} \lambda_q = 1 && (\pi_0) \\
& \lambda_q \geq 0, \mu_r \geq 0
\end{aligned}$$

dove π e π_0 sono le variabili duali associate ai vincoli.

3.5.1 Generazione di Colonne (Pricing Problem)

Il RMP viene risolto in modo iterativo. L'algoritmo Simplex richiede l'introduzione di una nuova colonna se il suo costo ridotto è negativo. Il costo ridotto per un punto estremo $\mathbf{v}_q$ è:

$$\bar{c}_q = \mathbf{c}^T \mathbf{v}_q - \pi^T A \mathbf{v}_q - \pi_0$$

Si cerca la colonna (punto o direzione estrema) con il costo ridotto più negativo risolvendo il **Sottoproblema** (o *Pricing Problem* - PP):

$$\min_{\mathbf{x} \in X} \quad (\mathbf{c}^T - \pi^T A)\mathbf{x}$$

Si noti che $\mathbf{x} \in X$ (solo vincoli facili) e non c'è il termine costante π_0. Il valore ottimale del PP, z^*_{PP}, determina l'azione successiva.

Risultati del Pricing Problem (PP)

- Se $z^*_{PP} = -\infty$: Il PP è illimitato inferiormente. È stata trovata una **direzione estrema** $\mathbf{x}^*$ con costo ridotto negativo. Si aggiunge una nuova variabile μ_r (colonna) al RMP.
- Se $-\infty < z^*_{PP} < \pi_0$: Il PP è limitato ed è stato trovato un **punto estremo** $\mathbf{x}^*$ con costo ridotto negativo. Si aggiunge una nuova variabile λ_q (colonna) al RMP.
- Se $z^*_{PP} \geq \pi_0$: Non c'è alcun punto o direzione utile da aggiungere. L'algoritmo **si ferma**.

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 ambienti non make-to-stock, poiché la propagazione dei fabbisogni lungo

la distinta base può generare **amplificazione della variabilità**. In questo contesto si colloca l'evoluzione dalla logica MRP ai sistemi MRPII ed ERP, come risposta al problema del lot-sizing multilivello.

La logica **MRP** si fonda su un'ipotesi 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 dall'MPS. Gli ordini pianificati non sono esecutivi e il processo è soggetto al fenomeno del **nervosismo**, per cui piccole variazioni dell'MPS possono produrre grandi variazioni negli ordini, anche per effetto di bordo dovuto alla ripianificazione rolling horizon.

A livello di shop floor, la **Factory Physics** mette in relazione throughput, flow time e WIP tramite la **legge di Little** e mostra come la variabilità e l'elevata utilizzazione aumentino i tempi di attraversamento. L'approccio **Just-In-Time (Toyota)** mira a ridurre la variabilità alla fonte mediante produzione livellata, controllo **pull** del WIP e riduzione dei tempi di setup, evidenziando il legame strutturale tra variabilità, livelli di magazzino e prestazioni del sistema produttivo.

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.

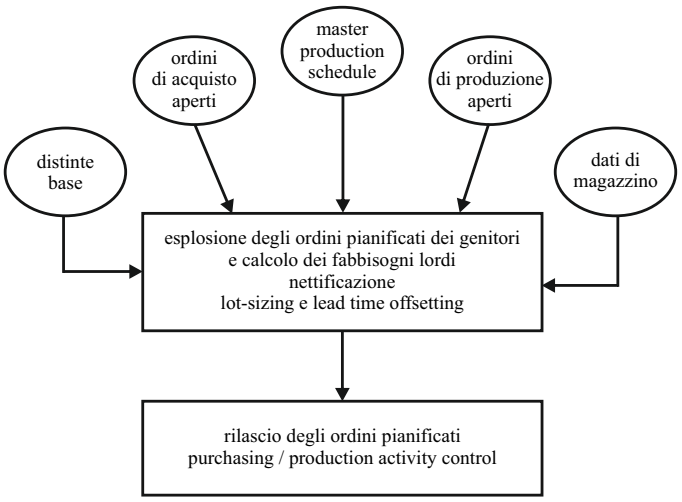
4.2 Evoluzione dai modelli MRP ai sistemi ERP

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:

- **RCCP** (Rough Cut Capacity Planning).
- **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

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. Infine, 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 pianificazione della capacità

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**.

4.7 Factory Physics e legge di Little

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:

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

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

4.8 Ruolo della variabilità nel flusso dei materiali

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:

$$\boxed{L = \lambda\,(W_q + t_s)}$$

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:

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

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.

4.9 Buffering Law e produzione livellata

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.

4.10 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*.

4.11 Toyota Goal Chasing

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:

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

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:

$$\boxed{\frac{kN_j}{Q}}$$

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:

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

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.

4.12 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:

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

Inoltre, il periodo di rotazione deve soddisfare il vincolo:

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

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:

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

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 Manca

5.1 Obiettivo: Massimizzazione del Profitto

L'obiettivo di un'impresa è tipicamente la massimizzazione del profitto, Π, definito come la differenza tra Ricavo Totale $R(q)$ e Costo Totale $C(q)$:

$$\Pi(q) = R(q) - C(q) = p(q)q - C(q)$$

dove $p(q)$ è la funzione di domanda inversa (prezzo in funzione della quantità) e q è la quantità prodotta/venduta.

Condizione di Ottimalità Per massimizzare il profitto in termini di quantità q, la condizione del primo ordine è:

$$\frac{d\Pi}{dq} = \frac{dR}{dq} - \frac{dC}{dq} = 0 \quad \Longrightarrow \quad MR = MC$$

Il **Ricavo Marginale (MR)** deve eguagliare il **Costo Marginale (MC)**.

5.1.1 Ricavo Marginale ed Elasticità della Domanda

Il Ricavo Marginale si esprime come:

$$MR = \frac{dR}{dq} = p + q\frac{dp}{dq}$$

L'**Elasticità della Domanda rispetto al Prezzo** (η) è definita come la variazione percentuale della quantità domandata rispetto alla variazione percentuale del prezzo:

$$\eta = \frac{dq/q}{dp/p} = \frac{dq}{dp}\frac{p}{q}$$

Poiché la domanda è inversamente proporzionale al prezzo, l'elasticità è negativa ($\eta < 0$).

Relazione tra MR ed Elasticità Il Ricavo Marginale può essere espresso in termini di prezzo ed elasticità, poiché $q\frac{dp}{dq} = p\frac{q}{p}\frac{dp}{dq} = p\frac{1}{\eta}$:

$$MR = p\left(1 + \frac{1}{\eta}\right)$$

L'ottimo $MR = MC$ implica la **regola di prezzo ottimale** (o Indice di Lerner):

$$\frac{p^* - MC}{p^*} = -\frac{1}{\eta}$$

Un'impresa con un maggiore potere di mercato (elasticità η più vicina a zero in valore assoluto) può applicare un *mark-up* superiore al costo marginale.

5.2 Discriminazione di Prezzo

La discriminazione di prezzo si verifica quando l'impresa applica prezzi diversi a diversi clienti o gruppi di clienti per lo stesso prodotto.

Discriminazione di Terzo Grado (Segmentazione del Mercato) Se il mercato può essere segmentato in due o più segmenti (es. studenti e adulti, residenti e turisti), l'impresa deve ottimizzare il prezzo per ciascun segmento i:

$$\frac{p_i - MC}{p_i} = -\frac{1}{\eta_i}$$

L'impresa applicherà un prezzo più alto al segmento con la **domanda meno elastica** (elasticità η_i più vicina a zero). La condizione di ottimalità richiede che i Ricavi Marginali per tutti i mercati eguaglino il Costo Marginale:

$$MR_1 = MR_2 = \cdots = MC$$

5.3 Modelli di Scelta Discreta (Discrete Choice Models - DCM)

I DCM si concentrano sul comportamento del cliente, in contrasto con l'approccio microeconomico che usa la curva di domanda aggregata. L'obiettivo è rappresentare ciò che la gente fa, basandosi sull'ipotesi che i clienti scelgano l'alternativa che massimizza la loro utilità.

Funzione di Utilità Random Sia $\mathcal{C} = \{0, 1, \ldots, n\}$ l'insieme delle alternative, dove 0 è l'alternativa di *no-purchase* (non acquistare). L'utilità dell'alternativa j per l'individuo i è:

$$U_{ij} = V_{ij} + \epsilon_{ij}$$

dove:

- V_{ij} è la **componente deterministica** (o utilità media), che dipende dalle caratteristiche del cliente i e del prodotto j (es. prezzo, qualità).
- ϵ_{ij} è la **componente stocastica** (o errore), che rappresenta l'ignoranza del modellatore, le preferenze intrinseche non osservabili, o l'errore di misurazione.

Probabilità di Scelta La probabilità che l'individuo i scelga l'alternativa j è:

$$P_{ij} = \mathbb{P}\{U_{ij} > U_{ik}, \quad \forall k \in \mathcal{C}, k \neq j\}$$

Questa probabilità dipende dalla distribuzione delle componenti stocastiche ϵ.

5.3.1 Il Modello Logit

Il modello Logit deriva dall'assunzione che gli errori stocastici ϵ_{ij} siano **indipendenti e identicamente distribuiti (IID)** secondo una **distribuzione di Gumbel** (o di Valore Estremo di Tipo I). La probabilità di scelta P_j (omettendo l'indice i per omogeneità) è data da:

$$P_j = \frac{e^{V_j}}{\sum_{k \in \mathcal{C}} e^{V_k}}$$

Proprietà della Scelta Irrilevante (IIA) Una conseguenza fondamentale dell'assunzione IID (e quindi del modello Logit) è la **Proprietà dell'Indipendenza dalle Alternative Irrilevanti (IIA - Independence from Irrelevant Alternatives)**.

$$\frac{P_j}{P_k} = \frac{e^{V_j}}{e^{V_k}} = e^{V_j - V_k}$$

Il rapporto tra le probabilità di scelta di due alternative j e k dipende **solo** dalle utilità deterministiche V_j e V_k, e non da qualsiasi altra alternativa presente nell'insieme di scelta $\mathcal{C}$.

Fallimento della Proprietà IIA La proprietà IIA fallisce in presenza di **sostituti vicini** (es. il paradosso del bus rosso/blu). Se le componenti stocastiche sono correlate tra alternative simili, il modello Logit non è appropriato.

5.3.2 Modelli Generalizzati e Distribuzioni di Valore Estremo

Per superare i limiti dell'IIA, si ricorre a modelli più generali, come il **Nested Logit** o il **Mixed Logit** (che permettono correlazioni tra gli errori).

Distribuzioni di Valore Estremo Generalizzate (GEV) Le distribuzioni di valore estremo generalizzate $H_\xi(x)$ sono una famiglia di distribuzioni che include:

$$H_\xi(x) = \exp\left\{-(1+\xi x)^{-1/\xi}\right\}, \quad \xi \neq 0$$

con l'aggiunta di parametri di posizione $\mu \in \mathbb{R}$ e scala $\sigma > 0$: $H_{\xi,\mu,\sigma}(x) = H_\xi\left(\frac{x-\mu}{\sigma}\right)$.

- Se $\xi > 0$: **Distribuzione di Fréchet**. Ha una coda pesante (*heavy tail*) e un decadimento lento.
- Se $\xi = 0$: **Distribuzione di Gumbel**. Ottenuta come limite per $\xi \to 0$. È fondamentale per il modello Logit.
- Se $\xi < 0$: **Distribuzione di Weibull**. Ha un estremo destro finito.

La distribuzione di Gumbel è quella che, quando usata come distribuzione per gli errori ϵ, genera il modello Logit.

6 CAP 6 –Pricing

7 CAP 7 –Il principio della programmazione dinamica

7.1 Il principio della programmazione dinamica

Definizione e Idea Centrale La **Programmazione Dinamica (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 gamma di algoritmi di ottimizzazione. In sostanza, è un principio per generare algoritmi di ottimizzazione. L'idea centrale della DP si basa sulla **scomposizione** di un problema di decisione dinamico a stadi multipli in una sequenza di problemi a stadio singolo più semplici. Le decisioni si prendono in base alle informazioni che arrivano in momenti successivi. In questo principio, i problemi si decompongono a stadio singolo, andando a fare tagli rispetto al tempo e non rispetto agli scenari. La DP è anche alla base del *reinforcement learning*.

Requisiti di Struttura La DP non è un approccio universale, poiché la sua applicazione richiede una **struttura specifica** nel modello di sistema, che deve essere **Markoviano**. Deve esserci il concetto di "Stato". La DP può essere applicata a una vasta gamma di problemi:

- **Problemi Continui e Discreti**: la caratteristica discreta/continua può riferirsi a variabili di stato, variabili di decisione o alla rappresentazione del tempo (si guardano solo tempi discreti).
- **Problemi Deterministici e Stocastici**.
- **Problemi a Orizzonte Finito e Infinito**.

7.2 Implementazione della programmazione dinamica

Problemi di Decisione Dinamici e Tempo Discreto Trattiamo **modelli a tempo discreto**. Non ha senso considerare problemi a tempo continuo. Le convenzioni temporali adottate sono:

- ◇ **Istanti di tempo** ($t = 0, 1, 2, \dots$): si osserva lo stato del sistema e si prende una decisione.
- ◇ **Intervalli di tempo** ($t = 1, 2, \dots$): il tempo che intercorre tra gli istanti $t-1$ e t. Il sistema evolve in questo intervallo, e un nuovo stato viene raggiunto all'istante t.

Le decisioni sono prese prima di osservare la realizzazione dei fattori di rischio.

Dinamiche del Sistema: Equazione di Transizione di Stato Le dinamiche del sistema sono rappresentate dall'equazione di transizione di stato:

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

dove:

- g_{t+1} è la **funzione di transizione di stato** sull'intervallo di tempo $t+1$.
- s_t è il vettore delle **variabili di stato** all'istante t, che riassume tutte le informazioni necessarie dall'evoluzione passata del sistema.
- x_t è il vettore delle **variabili di decisione** (o di controllo) all'istante t.
- ξ_{t+1} è un **fattore esogeno** (fattore di rischio) la cui realizzazione durante l'intervallo $t+1$ influenza la transizione al nuovo stato.

Tipi di Variabili di Stato È utile distinguere diverse tipologie di variabili di stato:

- ◦ **Variabili di stato fisico**: descrivono lo stato di risorse fisiche o finanziarie. Sono direttamente influenzate dalle nostre decisioni.
- ◦ **Variabili di stato informativo**: informazioni rilevanti (es. prezzo di un'azione) che non sono influenzate dall'attività di un singolo decisore in mercati liquidi.
- ◦ **Variabili di stato di credenza (*Belief state*)**: rilevanti in problemi affetti da incertezza sull'incertezza. I parametri del modello (o di una distribuzione di probabilità) diventano variabili di stato.

7.2.1 Politiche di Decisione (Policy)

Le decisioni x_t sono prese all'istante t, dopo aver osservato lo stato s_t. La politica deve essere **non anticipativa** perché esclude la previsione del futuro, a meno che il problema non sia deterministico. Una politica di decisione implementabile è detta non-anticipativa. Una politica di decisione *closed-loop* (a **feedback**) e non-anticipativa è espressa nella forma:

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

dove $\mu_t(\cdot)$ è una funzione che mappa lo stato s_t in una decisione ammissibile x_t.

7.3 Modelli della programmazione dinamica

Funzione Obiettivo L'obiettivo generale è ottimizzare una funzione obiettivo sull'orizzonte di pianificazione. La DP si presta a problemi in cui la funzione obiettivo è **additiva nel tempo**, cioè è la somma dei costi sostenuti (o dei premi guadagnati) in ogni istante di tempo.

Dipendenza nel Processo dei Dati (ξ_t) I vettori casuali nel processo dei dati possono avere diversi gradi di dipendenza reciproca:

- ◦ **Indipendenza tra gli stadi (Caso facile)**: gli elementi del processo dei dati sono mutualmente indipendenti (il processo non ha memoria).

- **Processo Markoviano (Caso abbastanza facile)**: ξ_{t+1} dipende solo dall'ultima realizzazione ξ_t dei fattori di rischio (Modello Markoviano del primo ordine).
- **Caso più generale**: ξ_{t+1} dipende dall'intera storia $\xi_{[t]}$ osservata fino al tempo t.

7.3.1 Problemi a Orizzonte Finito Scontato

Un problema stocastico con orizzonte finito T può essere espresso come:

$$\text{opt}_{\mu\in\mathcal{M}} E_0 \left[\sum_{t=0}^{T-1} \gamma^t f_t(s_t, \mu_t(s_t)) + \gamma^T F_T(s_T) \right]$$

dove:

* $f_t(s_t, x_t)$ è il **Contributo immediato** (costo o premio).
* $F_T(s_T)$ è il **Costo/premio terminale**, che dipende solo dallo stato finale s_T.
* $\gamma \in (0, 1)$ è il **fattore di sconto**, comune nelle applicazioni finanziarie. Per l'orizzonte infinito, è necessario per la convergenza della somma.

8 CAP 8 –Programmazione dinamica numerica per stati discreti

8.1 Programmazione dinamica approssimativa e apprendimento per rinforzo per stati discreti

Stima dei Contributi e Reinforcement Learning (RL) In alcuni casi, la determinazione del contributo immediato $f_t(s_t, x_t)$ tramite l'aspettativa condizionale $E_t[h(s_t, x_t, \xi_{t+1})]$ può essere difficile da calcolare o può mancare un modello sensato. In tali situazioni, l'unica possibilità è osservare un campione di contributi casuali, o tramite **simulazione Monte Carlo** o **sperimentazione *online***, come è tipico nell'apprendimento per rinforzo (*reinforcement learning*).

8.2 Implementazione della programmazione dinamica

8.3 Allocazione discreta delle risorse: il problema dello zaino (Knapsack Problem)

Il problema dello zaino richiede di selezionare un sottoinsieme di oggetti di massimo valore totale, soggetti a un vincolo di budget (capacità). Il problema è deterministico. Introduciamo variabili di decisione binarie x_k per modellare la selezione di ogni oggetto:

$$x_k = \begin{cases} 1 & \text{se l'oggetto } k \text{ è selezionato,} \\ 0 & \text{altrimenti.} \end{cases}$$

Il problema può essere formulato come un programma lineare binario puro:

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

Si assume un'allocazione discreta del budget, poiché la selezione di un oggetto è una decisione "tutto o niente". Il problema non è intrinsecamente dinamico, ma può essere riformulato come un problema sequenziale di allocazione delle risorse introducendo un indice temporale discreto fittizio k, corrispondente agli oggetti. Per ogni indice $k = 1, \ldots, n$, dobbiamo decidere se includere l'oggetto k nel sottoinsieme. La

decisione dell'ultimo oggetto ($k = n$) è banale, poiché si considera solo il sottoinsieme $\{n\}$ dato il budget residuo. La selezione degli oggetti successivi è influenzata dalle decisioni passate solo attraverso il budget residuo.

Variabile di Stato e Transizione La variabile di stato naturale al passo k è il budget disponibile s_k prima di selezionare l'oggetto k. La variabile di decisione è x_k. L'equazione di transizione di stato è:

$$s_{k+1} = s_k - w_k x_k, \quad k = 1, \ldots, n,$$

con condizione iniziale $s_1 = B$. Assumendo che i requisiti di risorsa w_k siano interi, anche la variabile di stato sarà un numero intero.

Funzione di Valore e Ricorrenza DP Definiamo la funzione di valore $V_k(s)$ come il profitto derivante dalla selezione ottimale del sottoinsieme all'interno dell'insieme di oggetti $\{k, k+1, \ldots, n\}$, quando il budget residuo è s. La ricorsione della DP è:

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

Questa equazione stabilisce che al passo k, si considera l'oggetto k:

- Se il suo peso w_k non rientra nel budget residuo s ($s < w_k$), l'oggetto viene escluso. La funzione di valore rimane $V_{k+1}(s)$.
- Altrimenti, si confrontano due alternative: 1) Includere l'oggetto k, guadagnando v_k e allocando in modo ottimale il budget aggiornato $s - w_k$ agli oggetti successivi, oppure 2) Non includere l'oggetto k, allocando tutto il budget residuo s in modo ottimale agli oggetti successivi.

La condizione terminale per l'ultimo oggetto $k = n$ è:

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

Complessità Computazionale La complessità computazionale di questa procedura, assumendo dati interi, è $O(nB)$, che non è polinomiale ma pseudo-polinomiale. L'implicazione pratica è che, quando B è un numero grande, si deve costruire una tabella enorme, rendendo l'algoritmo poco efficiente.

8.4 Problema dello zaino: implementazione MATLAB

Struttura della Soluzione La funzione `DPKnapsack` riceve i vettori `value` e `weight` e la capacità scalare `capacity`. Restituisce il sottoinsieme ottimale di oggetti (vettore binario `X`) e il valore totale (`reward`).

- La funzione di valore è memorizzata nella matrice `valueTable`, con n colonne (oggetti) e $B+1$ righe (stati $s = 0, 1, \ldots, B$).
- Le decisioni ottimali per ogni stato sono raccolte nella matrice `decisionTable` (funzione di politica in forma tabellare).
- Il ciclo `for` più esterno implementa la ricorsione all'indietro della DP.
- Dopo il ciclo principale, la soluzione ottimale viene costruita procedendo in avanti rispetto agli oggetti, prendendo decisioni e aggiornando la variabile di stato (`resCapacity`) di conseguenza.

8.5 Allocazione continua del budget

Consideriamo una versione continua del problema di allocazione delle risorse, dove un budget B deve essere allocato a n attività. L'allocazione all'attività k è una variabile di decisione continua $x_k \geq 0$. Il contributo al profitto dipende dall'allocazione delle risorse attraverso una funzione crescente e concava $f_k(\cdot)$:

$$\max \sum_{k=1}^{n} f_k(x_k) \quad \text{s.t.} \quad \sum_{k=1}^{n} x_k \leq B \quad x_k \geq 0 \quad k = 1, \ldots, n.$$

Se le funzioni di profitto sono concave, il problema è relativamente facile da risolvere. Se la funzione è concava (come $f_k(x) = \sqrt{x}$), si assegna a tutti una porzione equa.

Formulazione DP per il Continuo Il problema può essere riformulato in un framework DP associando un indice temporale fittizio $k = 1, \ldots, n$ a ciascuna attività. Sia $V_k(s)$ il profitto ottimale derivante dall'allocazione di un budget residuo s (la variabile di stato) alle attività nell'insieme $\{k, k+1, \ldots, n\}$. L'equazione di transizione di stato è:

$$s_{k+1} = s_k - x_k,$$

con condizione iniziale $s_1 = B$. Le funzioni di valore soddisfano le equazioni di ottimalità:

$$V_k(s_k) = \max_{0 \leq x_k \leq s_k} \{f_k(x_k) + V_{k+1}(s_k - x_k)\}$$

con condizione terminale:

$$V_n(s_n) = \max_{0 \leq x_n \leq s_n} f_n(x_n) = f_n(s_n).$$

Nel caso continuo, la funzione di valore $V_t(s)$ appartiene a uno spazio infinito-dimensionale di funzioni. Poiché possiamo valutare $V_t(s)$ solo su un insieme finito di stati, è necessario ricorrere a un metodo di approssimazione o interpolazione (come l'interpolazione polinomiale o le spline cubiche) per trovare i valori di stato al di fuori della griglia.

8.6 Interludio: interpolazione di funzioni con spline cubiche in MATLAB

È noto che l'interpolazione polinomiale soffre di oscillazioni inaccettabili. Una bella funzione, concava e monotonicamente crescente, può essere approssimata da una funzione non monotona, cosa che creerebbe problemi nelle procedure di ottimizzazione. Un'alternativa standard è ricorrere a una funzione polinomiale a tratti di ordine inferiore, dove ogni tratto è un polinomio associato a un singolo sotto-intervallo della griglia. Una scelta comune è utilizzare le **spline cubiche**, facilmente realizzabili in MATLAB tramite una coppia di funzioni:

- `spline`: restituisce un oggetto spline, basato sui valori della griglia.
- `ppval`: valuta la spline su punti arbitrari, al di fuori della griglia dei dati.

8.7 Soluzione del problema di allocazione continua del budget con DP numerica

Si utilizza una spline cubica per approssimare la funzione di valore del problema di allocazione continua del budget. Si imposta una griglia uniforme per l'intervallo $[0, B]$, replicata per ogni stadio. La griglia include $m+1$ valori di stato, con passo di discretizzazione $\delta s = B/m$. Dobbiamo risolvere un sottoproblema della forma (3) per ogni punto della griglia, cosa che può essere eseguita dalla funzione MATLAB `fminbnd`. Si interpola al di fuori della griglia con spline cubiche per approssimare i valori di $V_k(s)$ per un budget residuo s arbitrario. La politica ottimale $x_t^* = \mu_t^*(s_t)$ è implicita nella sequenza delle funzioni di valore.

8.8 Dimensionamento stocastico del lotto (Stochastic Lot Sizing)

Il termine di costo immediato dipende dalla realizzazione del fattore di rischio ξ_{t+1} (domanda d_{t+1}) durante il periodo di tempo $t+1$ dopo aver preso la decisione x_t. Ciò implica che la ricorsione DP ha un costo immediato stocastico della forma $h_t(s_t, x_t, \xi_{t+1})$. La ricorsione DP risultante (nell'esempio di Bertsekas, Vol. 1) è:

$$V_t(I_t) = \min_{x_t \in X(I_t)} E_{d_{t+1}} \left[cx_t + \beta \left(I_t + x_t - d_{t+1}\right)^2 + V_{t+1}\left(\max\left\{0, I_t + x_t - d_{t+1}\right\}\right)\right]$$

dove $I_t \in \{0, 1, 2, \ldots, I_{\max}\}$ è lo stato (livello di inventario) al tempo t. L'incertezza è modellata da una funzione di massa di probabilità (vettore di probabilità π_k) per ogni possibile valore di domanda $k = 0, 1, 2, \ldots, d_{\max}$. La funzione MATLAB `MakePolicy` calcola le funzioni di valore e la politica ottima in forma tabellare.

8.9 Sfruttare la struttura: usare il percorso minimo per il dimensionamento deterministico del lotto

Nel dimensionamento deterministico del lotto (assumendo solo costi di ordine fisso ϕ e costi di mantenimento dell'inventario h, con inventari iniziali e terminali pari a zero), si può sfruttare la struttura del problema. La condizione di Wagner–Whitin implica che al tempo t si dovrebbero considerare solo le seguenti possibilità di ordine:

$$x_t \in \left\{ 0, d_{t+1}, (d_{t+1} + d_{t+2}), (d_{t+1} + d_{t+2} + d_{t+3}), \ldots, \sum_{\tau=t+1}^{T} d_\tau \right\}$$

Grazie a questa proprietà, il problema a articolo singolo può essere riformulato come un **problema del percorso minimo (shortest path)** su una rete piuttosto piccola, dove gli archi rappresentano il numero di intervalli di tempo coperti dal prossimo ordine.

8.10 Dimensionamento stocastico del lotto: politiche S e (s, S)

Nel dimensionamento stocastico del lotto (senza penalità per vendite perse, ma con costi di mantenimento h e di stockout/arretrato b), si considera una penalità convessa: $q(s) = h\max\{0, s\} + b\max\{0, -s\}$. La ricorsione DP può essere scritta come:

$$V_t(I_t) = \min_{x_t \geq 0} \left\{ cx_t + H(I_t + x_t) + E\left[V_{t+1}(I_t + x_t - d_{t+1})\right]\right\}$$

dove $y_t = I_t + x_t$ è l'inventario disponibile dopo l'ordine e $H(y_t)$ è la funzione del costo di un periodo:

$$H(y_t) = E\left[q(y_t - d_{t+1})\right] = hE\max\{0, y_t - d_{t+1}\} + bE\max\{0, d_{t+1} - y_t\}$$

In assenza di costi di ordine fissi, la politica ottimale è una **politica base-stock** (o *order-up-to*):

$$x_t^* = \mu_t^*(I_t) = \begin{cases} S_t - I_t & \text{se } I_t < S_t \\ 0 & \text{se } I_t \geq S_t \end{cases}$$

dove S_t è il livello di inventario obiettivo (target inventory level). Se si includono costi di ordine fissi, la politica ottimale diventa una **politica (s, S)**.

8.11 Le maledizioni della programmazione dinamica (The curses of DP)

La DP è un principio potente e flessibile, ma presenta alcune importanti limitazioni:

- **La maledizione della dimensionalità dello stato (Curse of state dimensionality)**: è necessaria la funzione di valore per ogni elemento nello spazio degli stati. Questo non è fattibile per spazi di stato enormi.
- **La maledizione dell'ottimizzazione (Curse of optimization)**: la DP decompone un problema intrattabile a stadi multipli in una sequenza di sottoproblemi a stadio singolo. Tuttavia, anche i sottoproblemi a stadio singolo possono essere molto difficili da risolvere.
- **La maledizione dell'aspettativa (Curse of expectation)**: se i fattori di rischio ξ_t sono rappresentati da variabili casuali continue, l'aspettativa richiede il calcolo di un difficile integrale multidimensionale; è necessaria una strategia di discretizzazione.
- **La maledizione della modellazione (Curse of modeling)**: il sistema stesso può essere così complesso da rendere impossibile trovare un modello esplicito delle transizioni di stato.

9 CAP 9 – Modelli della programmazione dinamica

9.1 Modellazione per la DP

Il principio della DP è un concetto flessibile per la scomposizione di un problema decisionale a stadi multipli in una sequenza di problemi a stadio singolo, bilanciando il contributo immediato e i contributi attesi dalle decisioni future. Ciò dipende dalla funzione di valore definita dalla ricorsione DP:

$$V_t(s_t) = \text{opt}_{x_t \in X(s_t)} \{f_t(s_t, x_t) + \gamma E[V_{t+1}(s_{t+1})|s_t, x_t]\} \quad (1)$$

Questa è solo una possibile forma di ricorsione DP. Alcune situazioni prevedono lo scambio del valore atteso con l'ottimizzazione. Un rifacimento più radicale viene talvolta adottato, in cui si scambiano aspettativa e ottimizzazione. Questo può verificarsi a causa della struttura informativa del problema o può essere il risultato di manipolazioni volte a evitare la soluzione di un difficile sottoproblema di ottimizzazione stocastica.

Stati Post-Decisione e Q-Learning Si ottengono introducendo gli **stati post-decisione** (*post-decision states*). Un caso ben noto si verifica nel **Q-learning**, una forma di *reinforcement learning* (RL) dove la funzione di valore dello stato $V(s)$ è sostituita dai **Q-fattori** $Q(s, x)$, che rappresentano il valore delle coppie stato-azione. Un grosso vantaggio è che non è necessario conoscere le distribuzioni di probabilità. Il *reinforcement learning* è una **programmazione dinamica *model free*** (senza modello), un algoritmo puramente statistico che **NON conosce** le probabilità di transizione $\Pi_{i,j}(\cdot)$.

Descrizione dello Stato e Modello Markoviano Anche trovare una descrizione adeguata dello stato del sistema può richiedere un'attenta modellazione. Ad esempio, potrebbe essere necessaria una ridefinizione dello spazio degli stati per eliminare le dipendenze dal percorso (*path-dependencies*) e ottenere un modello Markoviano aumentato ed equivalente, al fine di rendere il problema trattabile con l'approccio DP. È naturale pensare agli stati e alle decisioni di controllo come scalari o vettori, ma possono anche consistere in oggetti diversi, come insiemi. La modellazione va di pari passo con la conoscenza algoritmica, quando si affronta un problema decisionale con la DP.

9.1.1 Esempi di Modellazione e Dipendenza dal Percorso

- **Opzione Call Europea**: Si può esercitare solo alla scadenza. Il *pay-off* è $\max(S_T - K, 0)$, dove K è lo *strike price*. Quello che conta è il prezzo adesso, non come ci si è arrivati, soprattutto se si ha un processo Markoviano.
- **Opzione Call Americana**: Si può esercitare in qualsiasi momento prima della scadenza. Quando bisogna esercitarla? Si sfrutta la programmazione dinamica, e il tempo che manca alla scadenza inciderà sulla scelta.

- **Opzione Call Asiatica**: Il *pay-off* è max(media dei prezzi osservati $- K, 0$). La media dei prezzi osservati precedentemente dipende dal percorso. Per eliminare questa dipendenza, non Markoviana, si **aggiunge una nuova variabile di stato aumentata** (tendenzialmente un integrale) rendendo il processo di nuovo Markoviano.

9.2 Processi Decisionali di Markov Finiti (Finite MDPs)

Il termine **Processo Decisionale di Markov (MDP)** è riservato a problemi caratterizzati da spazi discreti di stato e azione. Poiché stati e azioni sono discreti, possono essere enumerati e associati a numeri interi. Consideriamo solo MDPs finiti.

- **Spazio degli Stati**: $\mathcal{S} = \{1, ..., N\}$, dove N è la dimensione dello spazio.
- **Azioni/Decisioni**: $a \in \mathcal{A}_t(i)$ o $\mathcal{A}(i)$ denota l'insieme di azioni ammissibili nello stato i al tempo t.

L'obiettivo è tirare fuori le probabilità di transizione $\pi_{t+1}(i, a, j)$, che sono difficilmente calcolabili. In un MDP, le probabilità di transizione dipendono dall'azione selezionata:

$$\pi_{t+1}(i, a, j)$$

è la probabilità di una transizione dallo stato i allo stato j durante l'intervallo di tempo $t + 1$, dopo aver scelto l'azione a al tempo t.

9.2.1 Esempio: Controllo Stocastico dell'Inventario

Consideriamo il problema del controllo stocastico dell'inventario in cui la domanda può assumere valori in $\{0, 1, 2\}$ con probabilità $0.1, 0.7, 0.2$ rispettivamente. Il livello massimo di inventario è $I_t \leq 2$.

- **Spazio degli Stati**: $\mathcal{S} = \{0, 1, 2\}$.
- **Insiemi di Azioni Ammissibili** ($\mathcal{A}(i)$ è l'ammontare ordinato x_t): $\mathcal{A}(0) = \{0, 1, 2\}$, $\mathcal{A}(1) = \{0, 1\}$, $\mathcal{A}(2) = \{0\}$.

Le matrici di probabilità di transizione $\Pi(a)$, dove $\pi_{ij}(a)$ è la probabilità $\pi(i, a, j)$, sono (sulle righe lo stato corrente, sulle colonne lo stato futuro):

$$\Pi(0) = \begin{bmatrix} 0.2 & 0.7 & 0.1 \\ 0.9 & 0.1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad \Pi(1) = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 1 & 0 & 0 \\ \cdot & \cdot & \cdot \end{bmatrix}, \quad \Pi(2) = \begin{bmatrix} 1 & 0 & 0 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}.$$

(Per $a = 0$: non ordino nulla; per $a = 1$: ordino 1; per $a = 2$: ordino 2). Il punto è usato per le righe corrispondenti a stati in cui un'azione non è ammissibile. Ad esempio, l'azione $a = 2$ è ammissibile solo nello stato $i = 0$.

9.3 Ricorsioni della DP (DP Recursions)

Orizzonte Finito In un MDP, la funzione di valore a qualsiasi istante di tempo t si riduce a un vettore finito-dimensionale con componenti $V_t(i)$ e soddisfa:

$$V_t(i) = \text{opt}_{a \in \mathcal{A}(i)} \left\{ f_t(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi_{t+1}(i, a, j) V_{t+1}(j) \right\}, \quad i \in \mathcal{S}$$

Questo è il calcolo delle *Value Function* all'indietro.

Orizzonte Infinito Scontato Nel caso di orizzonte infinito scontato, sparisce il contributo del tempo e la funzione di valore è un punto fisso dell'operatore:

$$V(i) = \text{opt}_{a\in\mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j\in\mathcal{S}} \pi(i,a,j)V(j) \right\}, \quad i \in \mathcal{S}. \quad (3)$$

Se il fattore di sconto $\gamma < 1$ (fattore di contrazione), un punto fisso esiste ed è unico.

Contributo Immediato Stocastico Se il contributo immediato è stocastico e dipende dallo stato successivo j (dipendenza dalla realizzazione dei fattori di rischio), lo denotiamo come $h(i,a,j)$:

$$V(i) = \text{opt}_{a\in\mathcal{A}(i)} \sum_{j\in\mathcal{S}} \pi(i,a,j) \left\{ h(i,a,j) + \gamma V(j) \right\}, \quad i \in \mathcal{S}. \quad (4)$$

L'ottimizzazione è fuori, il valore atteso è dentro. Qui $h(i,a,j)+\gamma V(j)$ rappresenta il contributo immediato stocastico più il valore futuro scontato.

9.3.1 Esempio: Arresto Ottimale Ricorrente (*Recurrent Optimal Stopping*)

Consideriamo una catena lineare di stati $k = 1, 2, \ldots, n$, in cui lo stato può solo muoversi a destra o può essere resettato, ottenendo una ricompensa (*reward*) e un riavvio del sistema. I *reward* R_k sono crescenti lungo la catena ($R_k < R_{k+1}$), creando un *tradeoff* tra raccogliere il *reward* immediato e attendere opportunità migliori.

- **Stato 1**: Non c'è alcuna decisione da prendere. Si rimane nello stato 1 con probabilità $\pi_{1,1}$ o si passa allo stato 2 con probabilità $\pi_{1,2}$.
- **Stati Intermedi** ($k = 2, \ldots, n-1$): Si sceglie tra due azioni: **Wait** (Aspettare) o **Reset** (Esercitare subito).
  - **Wait**: Nessuna ricompensa, si rimane in k con probabilità $\pi_{k,k}$ o si passa a $k+1$ con $\pi_{k,k+1}$.
  - **Reset**: Si ottiene la ricompensa immediata R_k e si torna allo stato 1 (transizione deterministica).
- **Stato Finale** (n): Si ottiene la massima ricompensa R_n e si torna allo stato 1.

I contributi immediati sono:

$$f(k, \text{wait}) = 0 \quad \text{e} \quad f(k, \text{reset}) = R_k.$$

Le equazioni DP per trovare il valore stazionario $V(i)$ sono (con fattore di sconto γ):

$$\begin{aligned} V(1) &= \gamma\pi_{1,1}V(1) + \gamma\pi_{1,2}V(2) \\ V(k) &= \max\left\{ R_k + \gamma V(1), \quad \gamma\pi_{k,k}V(k) + \gamma\pi_{k,k+1}V(k+1) \right\}, \quad k = 2, \ldots, n-1 \\ V(n) &= R_n + \gamma V(1) \end{aligned}$$

Questo sistema di equazioni lineari a tratti (a causa dell'operatore max) può essere risolto con metodi iterativi come la *Value Iteration*.

9.4 Valutazione della Politica e Q-fattori

Le difficoltà nell'affrontare gli MDPs derivano dalla dimensione dello spazio degli stati (curse of dimensionality) e dalla difficoltà o impossibilità di trovare l'intero set di probabilità di transizione $\pi(i,a,j)$ (curse of modeling). Un ingrediente per aggirare queste difficoltà è l'utilizzo del **campionamento Monte Carlo**. Un altro è riscrivere la ricorsione DP basandosi sui **Q-fattori** $Q(i,a)$, che misurano il valore di intraprendere l'azione a quando si è nello stato i.

Valutazione della Politica Data una politica stazionaria ammissibile μ che mappa ogni stato i in un'azione $a = \mu(i)$, la funzione di valore dello stato $V_\mu(i)$ può essere trovata risolvendo un sistema di equazioni lineari (senza ottimizzazione):

$$V_\mu(i) = f(i, \mu(i)) + \gamma \sum_{j \in \mathcal{S}} \pi(i, \mu(i), j) \cdot V_\mu(j), \quad i \in \mathcal{S}. \quad (5)$$

Questa equazione è fondamentale per i metodi numerici basati sulla *Policy Iteration*.

Definizione dei Q-fattori Il **Q-fattore per la politica stazionaria** μ è definito come:

$$Q_\mu(i, a) = f(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi(i, a, j) \cdot V_\mu(j). \quad (6)$$

L'idea è di applicare l'azione a nello stato corrente i, e poi seguire la politica μ.

Ricorsione DP con Q-fattori I **Q-fattori ottimali** sono definiti inserendo la funzione di valore ottimale $V(j)$ nella (6):

$$Q(i, a) = f(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi(i, a, j) V(j), \quad i \in \mathcal{S}, a \in \mathcal{A}(i). \quad (7)$$

Poiché $V(j) \equiv \text{opt}_{a \in \mathcal{A}(j)} Q(j, a)$, possiamo riscrivere la ricorsione DP in termini di Q-fattori:

$$Q(i, a) = f(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi(i, a, j) \left(\text{opt}_{\tilde{a} \in \mathcal{A}(j)} Q(j, \tilde{a}) \right), \quad i \in \mathcal{S}, a \in \mathcal{A}(i). \quad (8)$$

- **Svantaggio**: Invece di $V(i)$, si lavora con $Q(i, a)$, il che peggiora la maledizione della dimensionalità (*curse of dimensionality*).
- **Vantaggio**: Si **scambiano aspettativa e ottimizzazione**. È spesso più facile risolvere molti problemi di ottimizzazione semplici (deterministici) che un singolo problema grande (stocastico).

Inoltre, i Q-fattori possono essere appresi tramite campionamento statistico, aprendo la strada alla DP *model-free*.

9.5 Diverse Sfumature della DP Stocastica

La ricorsione DP di base (1) si basa sulle seguenti assunzioni:

1. Si osserva lo stato s_t al tempo t.
2. Si prende una decisione ammissibile $x_t \in X(s_t)$.
3. Si osserva un contributo immediato $f_t(s_t, x_t)$.
4. Si passa a un nuovo stato s_{t+1}, che dipende dai fattori di rischio ξ_{t+1}.

Per trovare $V_t(\cdot)$, si deve risolvere un problema di ottimizzazione stocastica con un'aspettativa impegnativa. Tuttavia, in alcuni casi la struttura informativa richiede di **scambiare aspettativa e ottimizzazione**.

9.5.1 Esempio: Dimensionamento del Lotto con Lookahead Limitato

Se all'istante t si ha la conoscenza perfetta della domanda d_{t+1}, il contributo di costo immediato è deterministico:

$$f_t(I_t, x_t, d_{t+1}) = hI_t + cx_t + \phi \cdot \delta(x_t) + q \cdot \max\{0, d_{t+1} - I_t - x_t\},$$

dove h è il costo di mantenimento, c e ϕ sono i costi di ordinazione (variabile e fisso) e q è la penalità per le vendite perse. Se prima si osserva la domanda d_{t+1} e poi si prende la decisione x_t, la ricorsione DP è:

$$V_t(I_t) = E_{d_{t+1}}\left[\min_{x_t \geq 0}\{f_t(I_t, x_t, d_{t+1}) + V_{t+1}(I_{t+1})\}\right].$$

L'aspettativa è fuori e la minimizzazione è dentro.

9.6 Variabili di Stato Post-Decisione

La ricorsione DP con operatori scambiati (aspettativa fuori, ottimizzazione dentro) presenta vantaggi: il problema di ottimizzazione interno è deterministico e l'aspettativa esterna può essere stimata tramite campionamento statistico. Questo può essere ottenuto riscrivendo le dinamiche del sistema basate sulle **variabili di stato post-decisione** (s_x^t). Lo stato s_x^t è uno stato intermedio osservato dopo che la decisione x_t è stata presa, ma prima che il fattore di rischio ξ_{t+1} sia realizzato.

Splitting della Transizione Si scompone l'equazione di transizione standard $s_{t+1} = g_{t+1}(s_t, x_t, \xi_{t+1})$ in due fasi:

$$s_x^t = g_1^t(s_t, x_t) \quad \text{(Stato post-decisione)} \quad (10)$$
$$s_{t+1} = g_2^{t+1}(s_x^t, \xi_{t+1}) \quad \text{(Transizione successiva)} \quad (11)$$

Nel dimensionamento del lotto, lo stato post-decisione è il livello di inventario I_x^t dopo l'ordine, ma prima della domanda: $I_x^t = I_t + x_t$.

Funzione di Valore Post-Decisione Si introduce il valore dello stato post-decisione $V_x^t(s_x^t)$:

$$V_x^t(s_x^t) = E\left[V_{t+1}(s_{t+1})|s_x^t\right]. \quad (12)$$

La ricorsione DP standard può essere riscritta come un problema di ottimizzazione deterministico:

$$V_t(s_t) = \text{opt}_{x_t \in X(s_t)}\left\{f_t(s_t, x_t) + \gamma V_x^t(s_x^t)\right\}. \quad (13)$$

Combinando (12) e (13), si ottiene nuovamente una ricorsione DP con lo scambio tra aspettativa e ottimizzazione (spostando l'indice indietro di un passo $t-1$):

$$V_x^{t-1}(s_x^{t-1}) = E\left[\text{opt}_{x_t \in X(s_t)}\left\{f_t(s_t, x_t) + \gamma V_x^t(s_x^t)\right\}|s_x^{t-1}\right]. \quad (14)$$

I Q-fattori rientrano naturalmente in questo *framework*, poiché la coppia stato-azione (i, a) può essere considerata uno stato post-decisione.

9.7 Aumento dello Stato nella Gestione dell'Inventario

Se il *lead time* di consegna ($LT \geq 1$) è un numero intero di intervalli di tempo, è necessario introdurre variabili di stato che tengano traccia di ciò che è stato ordinato in passato ed è ancora in transito. Sia $z_{t,\tau}$ l'ammontare che verrà consegnato τ intervalli di tempo dopo il tempo corrente t, dove $\tau = 0, 1, 2, \ldots, LT-1$.

- **Inventario disponibile**: $z_{t,0}$ è ciò che è immediatamente disponibile.
- **Transizione Inventario *on-hand***: $I_{t+1} = I_t + z_{t,0} - d_{t+1}$ (ignorando l'incertezza sulla domanda).
- **Transizione variabili di stato aggiuntive**: L'ammontare ordinato x_t al tempo t sarà disponibile LT intervalli dopo. Quindi:

  $$z_{t+1,LT-1} = x_t.$$

  Per $\tau < LT-1$, l'equazione di transizione è un semplice scorrimento temporale:

  $$z_{t+1,\tau} = z_{t,\tau+1}, \quad \tau = 0, 1, \ldots, LT-2. \quad (15)$$

Allo stesso modo, per gli articoli deperibili, è necessario descrivere l'inventario per età, introducendo un *array* di variabili di stato $I_{t,\tau}$ che rappresentano l'ammontare di inventario *on-hand* al tempo t con un'età di τ periodi di tempo.

9.8 Gestione delle Entrate (*Revenue Management*)

La gestione delle entrate (o *yield management*) mira a massimizzare le entrate derivanti dalla vendita di risorse deperibili.

- **Approcci**: basato sulla quantità (limitazione della disponibilità) e basato sul prezzo (prezzi dinamici).
- **Modello**: C unità di una singola risorsa (es. posti a sedere) allocate a n classi di tariffa $j = 1, \dots, n$, con prezzi $p_1 > p_2 > \cdots > p_n$.

Le caratteristiche essenziali da considerare sono:

1. **Comportamento del Cliente**: In un mercato perfettamente segmentato, ogni passeggero acquista una sola classe. Con le preferenze, è necessario un **modello di scelta**.
2. **Tempistica della Domanda**: I modelli sono "statici" se le richieste per le diverse classi non si intersecano nel tempo, e "dinamici" se le richieste sono intervallate nel tempo.

La politica decisionale può essere espressa in termini di **livelli di protezione** (quantità massima disponibile per classe) o **prezzi di offerta** (*bid-prices*, prezzo minimo di vendita di un posto).

9.8.1 Modello Statico con Segmentazione Perfetta della Domanda

Le domande per le classi sono indipendenti e avvengono sequenzialmente (es. D_n prima, poi D_{n-1}, ecc.). Gli stadi decisionali sono associati alle classi, indicizzate in ordine decrescente $j = n, n-1, \dots, 1$. Lo stato naturale è s_j, la capacità residua all'inizio dello stadio j, con stato iniziale $s_n = C$. Si cerca la sequenza di funzioni di valore $V_j(s_j)$ per trovare la massima entrata attesa $V_n(C)$. La condizione al contorno è $V_0(s_0) = 0$.

Ricorsione DP con Scambio di Operatori Per ogni classe j, si assume la seguente sequenza di eventi (non intuitiva, ma giustificabile):

1. Si osserva la domanda D_j per la classe j.
2. Si decide quante richieste accettare, x_j.
3. Si raccoglie l'entrata $p_j x_j$ e si procede allo stadio $j-1$ con $s_{j-1} = s_j - x_j$.

La ricorsione DP assume la forma scambiata:

$$V_j(s_j) = E_{D_j}\left[\max_{0 \le x_j \le \min\{s_j, D_j\}} \{p_j x_j + V_{j-1}(s_j - x_j)\}\right].$$

Con uno scorrimento di indici ($j+1 \to j$), si ha:

$$V_{j+1}(s) = E\left[\max_{0 \le x \le \min\{s, D_{j+1}\}} \{p_{j+1} x + V_j(s - x)\}\right]. \quad (16)$$

Valore Marginale Atteso della Capacità Introducendo il valore marginale atteso della capacità $\Delta V_j(s) = V_j(s) - V_j(s-1)$ (il costo opportunità di un posto), si può riscrivere l'equazione (16):

$$V_{j+1}(s) = V_j(s) + E\left[\max_{0 \le x \le \min\{s, D_{j+1}\}} \left(x \sum_{z=1} (p_{j+1} - \Delta V_j(s+1-z))\right)\right]. \quad (17)$$

10 CAP 10 – Programmazione dinamica numerica per stati discreti

10.1 Catene di Markov a Tempo Discreto (Discrete-time Markov chains)

Una catena di Markov a tempo discreto è un processo stocastico con una variabile di stato s_t, $t = 0, 1, 2, \ldots$ che può assumere un numero finito o infinito numerabile di valori $i \in \mathcal{S}$. La proprietà fondamentale di Markov è che lo stato futuro dipende solo dallo stato corrente e non dalla storia precedente (mancanza di memoria):

$$\mathbb{P}\{s_{t+1} = j | s_t = i, s_{t-1} = i_{t-1}, \ldots, s_0 = i_0\} = \mathbb{P}\{s_{t+1} = j | s_t = i\} = \pi_{t+1}(i, j).$$

Le probabilità di transizione $\pi_{t+1}(i, j)$ sono:

- La probabilità che il sistema transiti dallo stato i allo stato j nell'intervallo di tempo $t + 1$.
- Soddisfano $\pi_{t+1}(i, j) \geq 0$ e $\sum_{j \in \mathcal{S}} \pi_{t+1}(i, j) = 1$.

Se le probabilità di transizione non dipendono dal tempo, la catena è **stazionaria** o **omogenea**: $\pi_{t+1}(i, j) = \pi(i, j)$.

Notazione e Transizione (Caso Finito) Nel caso finito, lo spazio degli stati $\mathcal{S}$ ha dimensione N. La matrice di transizione Π è una matrice $N \times N$, dove $\Pi_{i,j} = \pi(i, j)$. La probabilità di transizione a due passi è data da:

$$\pi^{(2)}(i, j) = \sum_{k \in \mathcal{S}} \pi(i, k)\pi(k, j).$$

In forma matriciale, questo è il prodotto Π^2. La probabilità di transizione a k passi è data da Π^k.

Distribuzioni di Probabilità dello Stato La distribuzione di probabilità dello stato al tempo t è un vettore riga $p_t = [p_t(1), \ldots, p_t(N)]$, dove $p_t(i)$ è la probabilità che il sistema sia nello stato i. La dinamica di questa distribuzione è:

$$p_{t+1}(j) = \sum_{i \in \mathcal{S}} p_t(i)\pi(i, j).$$

In forma matriciale:

$$p_{t+1} = p_t \Pi.$$

Se una catena di Markov ha una **distribuzione stazionaria** π, essa soddisfa l'equazione:

$$\pi = \pi \Pi.$$

10.2 Problemi Decisionali di Markov (Markov Decision Problems - MDPs)

In un MDP, l'azione (decisione) $a \in \mathcal{A}(i)$ è scelta dall'insieme delle azioni ammissibili $\mathcal{A}(i)$, quando il sistema è nello stato i. Le probabilità di transizione dipendono dall'azione scelta:

$$\pi_{t+1}(i, a, j).$$

Per un MDP finito con orizzonte infinito scontato e una politica stazionaria (non dipendente dal tempo) $\mu : \mathcal{S} \to \mathcal{A}$, la funzione di valore $V(i)$ è il punto fisso dell'equazione DP:

$$V(i) = \max_{a \in \mathcal{A}(i)} \left\{ f(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi(i, a, j) V(j) \right\}, \quad i \in \mathcal{S}. \quad (9)$$

Definiamo l'operatore DP $\mathcal{T}$:

$$\mathcal{T}V(i) = \max_{a \in \mathcal{A}(i)} \left\{ f(i, a) + \gamma \sum_{j \in \mathcal{S}} \pi(i, a, j) V(j) \right\}. \quad (10)$$

La funzione di valore ottimale V^* è il punto fisso dell'operatore $\mathcal{T}$: $V^* = \mathcal{T}V^*$. Quando $\gamma < 1$, l'operatore $\mathcal{T}$ è una contrazione e ammette un unico punto fisso.

10.3 Algoritmi Iterativi per la Soluzione di MDPs a Orizzonte Infinito

Per risolvere l'equazione (9) (che è non lineare a causa dell'operatore max), si utilizzano metodi iterativi: l'Iterazione del Valore (Value Iteration - VI) e l'Iterazione della Politica (Policy Iteration - PI).

10.3.1 Iterazione del Valore (Value Iteration - VI)

VI utilizza l'operatore $\mathcal{T}$ in modo iterativo. Dato un vettore di valori iniziali $V^{(0)}$, l'iterazione è:

$$V^{(k+1)} = \mathcal{T}V^{(k)}, \quad k = 0, 1, 2, \ldots$$

La convergenza è garantita grazie alla proprietà di contrazione dell'operatore $\mathcal{T}$. VI è spesso lento a convergere ai valori precisi, ma tende a stabilire la politica ottimale μ^* in un numero minore di iterazioni rispetto a PI.

Passi della VI

1. **Inizializzazione**: Scegliere $V^{(0)}$ e porre $k = 0$.
2. **Iterazione del Valore**: $V^{(k+1)}(i) = \max_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) V^{(k)}(j) \right\}, \quad i \in \mathcal{S}.$
3. **Test di Convergenza**: Se $\max_i |V^{(k+1)}(i) - V^{(k)}(i)| < \epsilon$ (con ϵ piccolo), STOP.
4. **Miglioramento della Politica**: Se l'obiettivo è trovare anche la politica ottima, essa viene generata alla fine tramite:

$$\mu^*(i) = \arg\max_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) V^*(j) \right\}, \quad i \in \mathcal{S}.$$

10.3.2 Iterazione della Politica (Policy Iteration - PI)

PI alterna due fasi: Valutazione della Politica e Miglioramento della Politica. Converge in un numero finito di passi, ma ogni passo di Valutazione della Politica può essere costoso.

Passi della PI

1. **Inizializzazione**: Scegliere una politica ammissibile $\mu^{(0)}$ e porre $k = 0$.
2. **Valutazione della Politica** (Policy Evaluation): Calcolare la funzione di valore $V_{\mu^{(k)}}$ risolvendo il sistema di equazioni lineari (Eq. 5 nel DPChapter3):

$$V_{\mu^{(k)}}(i) = f(i, \mu^{(k)}(i)) + \gamma \sum_{j \in \mathcal{S}} \pi(i, \mu^{(k)}(i), j) V_{\mu^{(k)}}(j), \quad i \in \mathcal{S}. \quad (12)$$

   Questo sistema è di forma $V = c + \gamma \Pi_\mu V$, dove Π_μ è la matrice di transizione definita dalla politica μ, e può essere risolto come:

$$V_{\mu^{(k)}} = (I - \gamma \Pi_{\mu^{(k)}})^{-1} c_{\mu^{(k)}}.$$

3. **Miglioramento della Politica** (Policy Improvement): Trovare la nuova politica $\mu^{(k+1)}$ più avida (*greedy*) rispetto a $V_{\mu^{(k)}}$:

$$\mu^{(k+1)}(i) = \arg\max_{a \in \mathcal{A}(i)} \left\{ f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) V_{\mu^{(k)}}(j) \right\}, \quad i \in \mathcal{S}.$$

4. **Test di Convergenza**: Se $\mu^{(k+1)} = \mu^{(k)}$, STOP. Altrimenti, porre $k \leftarrow k + 1$ e tornare al passo 2.

10.3.3 Iterazione Generalizzata della Politica (Generalized Policy Iteration - GPI)

L'Iterazione Generalizzata della Politica (GPI) è un concetto astratto che generalizza VI e PI, dove l'algoritmo alterna i passi di Valutazione della Politica e Miglioramento della Politica senza richiedere la convergenza esatta della fase di valutazione.

- **VI come caso estremo**: La valutazione della politica è eseguita per una sola iterazione: $V^{(k+1)} \leftarrow \mathcal{T}V^{(k)}$.
- **PI come caso estremo**: La valutazione della politica è eseguita fino a convergenza (risolvendo il sistema lineare).

In un contesto numerico, la fase di Valutazione della Politica in PI può essere troncata dopo un certo numero di iterazioni di $\mathcal{T}_\mu$ prima di passare a una nuova politica, creando un continuum di approcci.

10.4 Relazione con il Reinforcement Learning (RL)

La differenza tra VI e PI diventa particolarmente rilevante nel contesto del *Reinforcement Learning* (RL), noto anche come DP *model-free* (senza conoscere le probabilità di transizione $\pi(i, a, j)$ e possibilmente i contributi immediati $f(i, a)$).

Q-Learning (Controparte di VI)

- Il **Q-learning** è la controparte RL dell'Iterazione del Valore.
- La funzione di valore $V(s)$ è sostituita dai Q-fattori $Q(s, a)$.
- È uno schema di apprendimento ***off-policy***, il che significa che una politica viene applicata (es. per esplorazione) per apprendere un'altra (la politica ottimale).
- L'aggiornamento dei Q-fattori è dato da:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]. \quad (13)$$

SARSA (Controparte di PI)

- Le controparti RL dell'Iterazione della Politica (PI) si basano su uno schema di apprendimento ***on-policy***, mirando ad apprendere il valore della stessa politica che si sta applicando.
- Un approccio ben noto è **SARSA** (State-Action-Reward-State-Action).
- L'aggiornamento SARSA è dato da:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \quad (14)$$

Limitazioni in RL In RL, non è possibile apprendere il valore di una politica in modo esatto (soprattutto se richiede costose simulazioni Monte Carlo o esperimenti *online*). È necessario eseguire un passo di miglioramento (*improving step*) prima o poi. Il modo preciso in cui ciò viene implementato lascia spazio a diverse varianti, portando a una serie di strategie RL.

11 CAP 11 – Programmazione dinamica approssimativa e apprendimento per rinforzo per stati discreti

11.1 Programmazione Dinamica Approssimativa (ADP)

Quando lo spazio degli stati $\mathcal{S}$ è troppo grande per essere risolto in forma tabellare con la Programmazione Dinamica (DP) (la **maledizione della dimensionalità**), dobbiamo ricorrere alla Programmazione Di-

namica Approssimativa (Approximate Dynamic Programming - ADP). La sua idea centrale è **approssimare la funzione di valore** $V(s)$ (o i Q-fattori $Q(s,a)$) utilizzando una funzione $\tilde{V}(s;\theta)$ (o $\tilde{Q}(s,a;\theta)$) che dipende da un vettore di parametri θ di dimensione molto più piccola.

$$V(s) \approx \tilde{V}(s;\theta) \quad \text{o} \quad Q(s,a) \approx \tilde{Q}(s,a;\theta).$$

L'uso di una funzione di valore approssimata trasforma la ricorsione DP in un problema di ottimizzazione parametrica stocastica.

Vantaggi dell'Approssimazione

- La dipendenza dal vettore di parametri θ consente di **generalizzare** i risultati a stati non visitati o stati che non sono nella griglia di risoluzione.
- La dimensione del problema è ridotta da $N = |\mathcal{S}|$ alla dimensione di θ.

Esempi di Funzioni di Approssimazione Le funzioni di approssimazione possono includere:

- **Funzioni lineari a tratti** (piecewise linear functions).
- **Polinomi** o **Spline** (come visto nel caso dell'allocazione continua del budget).
- **Reti Neurali** (Neuro-Dynamic Programming).

11.2 Apprendimento per Rinforzo (Reinforcement Learning - RL)

Il Reinforcement Learning (RL) si basa sull'idea di apprendere ***in situ*** (sul campo, attraverso simulazione o sperimentazione *online*) la funzione di valore o i Q-fattori, senza la necessità di un modello esplicito delle dinamiche di transizione $\pi(i,a,j)$ e dei contributi immediati $f(i,a)$.

DP Model-Free RL è una forma di Programmazione Dinamica ***model-free***.

- Potremmo **non avere un modello** del sistema (es. $\pi(i,a,j)$).
- Potremmo avere un modello ma questo è **troppo complicato** da applicare (es. DP con probabilità di transizione complesse).

In entrambi i casi, l'apprendimento avviene tramite **sperimentazione**, raccogliendo sequenze di (stato, azione, ricompensa, stato successivo): $(s_t, a_t, r_{t+1}, s_{t+1})$.

Equazioni di Bellman per Q-Fattori Nel contesto degli MDPs a orizzonte infinito, i Q-fattori ottimali $Q(i,a)$ soddisfano l'equazione di Bellman (come punto fisso):

$$Q(i,a) = f(i,a) + \gamma \sum_{j \in \mathcal{S}} \pi(i,a,j) \left(\max_{\tilde{a} \in \mathcal{A}(j)} Q(j,\tilde{a}) \right). \quad (1)$$

11.3 Q-Learning: L'Algoritmo

Il Q-learning è un algoritmo di *Temporal Difference* (TD) e rappresenta la controparte *model-free* dell'Iterazione del Valore (VI). Si basa sull'aggiornamento iterativo del Q-fattore $Q(s_t,a_t)$ (lo stato-azione corrente) utilizzando la ricompensa immediata r_{t+1} e una stima del valore futuro, $r_{t+1} + \gamma \max_a Q(s_{t+1},a)$.

Aggiornamento Q-Learning L'aggiornamento avviene in ogni passo, quando il sistema transita da (s_t, a_t) a s_{t+1} con ricompensa r_{t+1}:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]. \quad (2)$$

dove:

- α è il **tasso di apprendimento** (*learning rate*), $\alpha \in (0, 1)$. È un parametro che controlla quanto il nuovo valore stimato influenzi il valore corrente. Per garantire la convergenza, α deve diminuire nel tempo.
- r_{t+1} è la **ricompensa immediata** ottenuta con la transizione.
- $\gamma \max_a Q(s_{t+1}, a)$ è la **migliore stima del valore futuro** dello stato successivo s_{t+1}.

Vantaggi del Q-Learning

- È un approccio ***off-policy*** (aperto), che consente di apprendere i Q-fattori ottimali Q^* anche utilizzando una politica di esplorazione non ottimale, il che è essenziale per garantire la copertura dello spazio degli stati.
- Raggiunge la convergenza stocastica all'ottimo globale (se le condizioni su α sono soddisfatte e se tutti gli stati-azione sono visitati un numero infinito di volte).

11.4 Esempio: Arresto Ottimale Ricorrente (Risoluzione con Q-Learning)

Si usa il problema di arresto ottimale ricorrente (già visto) con 4 stati ($n = 4$) e *reward* R_k.

- **Stati**: $\mathcal{S} = \{1, 2, 3, 4\}$.
- **Azioni ammissibili**: $\mathcal{A}(1) = \{\text{wait}\}$; $\mathcal{A}(2) = \mathcal{A}(3) = \{\text{wait}, \text{reset}\}$; $\mathcal{A}(4) = \{\text{reset}\}$.
- **Transizioni/Ricompense**: Definite in una struttura `systemObj` (in MATLAB).

L'algoritmo `QLearning` riceve come input: l'oggetto di sistema, il tipo di ottimizzazione (*max* o *min*), il fattore di sconto γ, il numero di passi di apprendimento (*numSteps*), l'inizializzazione dei Q-fattori, il tasso di apprendimento α, e i parametri di esplorazione.

11.4.1 Esplorazione vs. Sfruttamento (*Exploitation vs. Exploration*)

Perché l'algoritmo converga, è necessario **esplorare** tutte le coppie stato-azione.

Politica ϵ-Greedy Una strategia comune per bilanciare sfruttamento e esplorazione è la **politica ϵ-greedy**.

- Con probabilità $1 - \epsilon$, l'agente **sfrutta** (sceglie l'azione a^* che massimizza il Q-fattore corrente: $\max_a Q(s, a)$).
- Con probabilità ϵ, l'agente **esplora** (sceglie un'azione casuale dall'insieme ammissibile $\mathcal{A}(s)$).

Se l'inizializzazione è insufficiente, è possibile che l'agente non provi mai un'azione che è effettivamente l'ottimale.

Esempio di Mancata Esplorazione Se i Q-fattori per l'azione *reset* negli stati 2 e 3 sono inizializzati a zero e i *reward* immediati sono tutti non negativi, l'azione *reset* potrebbe non essere mai provata, specialmente se la politica *wait* fornisce un buon valore iniziale. Ad esempio, con $Q(\cdot, \text{reset}) = 0$ per stati 2 e 3, e $\epsilon = 0$ (pura sfruttamento):

- Lo stato 4 è l'unico con azione *reset*.

- Lo stato 3, con $Q(3, \text{wait}) = 21.2052$ e $Q(3, \text{reset}) = 1.0000$, sceglie **wait**. L'azione *reset* non viene mai scelta.

Questo dimostra la necessità di bilanciare **sfruttamento** (scegliere l'azione migliore finora) e **esplorazione** (provare azioni sconosciute).

11.5 SARSA: State-Action-Reward-State-Action

SARSA è un altro algoritmo TD per apprendere i Q-fattori. A differenza di Q-learning, è un approccio ***on-policy***: apprende il valore della politica *che sta effettivamente utilizzando* per l'esplorazione.

Aggiornamento SARSA L'aggiornamento è dato da:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \quad (3)$$

- La differenza con Q-learning sta nel termine di valore futuro. In SARSA si usa $Q(s_{t+1}, a_{t+1})$, dove a_{t+1} è **l'azione effettivamente scelta** nello stato s_{t+1} (che sarà $a_{t+1} = \mu(s_{t+1})$), mentre Q-learning usa $\max_a Q(s_{t+1}, a)$ (l'azione ottimale).
- Questo rende SARSA **dipendente dalla politica di esplorazione**: se la politica di esplorazione è insufficiente, anche la politica appresa $Q(s, a)$ non sarà l'ottimale $Q^*(s, a)$.

11.6 Differenze tra Q-Learning e SARSA

- **Q-Learning (*Off-Policy*)**: Apprende il valore della **politica ottima** Q^* indipendentemente dalla politica di comportamento utilizzata (purché esplorativa). È un approccio più ambizioso.
- **SARSA (*On-Policy*)**: Apprende il valore della **politica di comportamento** utilizzata (es. la politica ϵ-greedy). È più cauto e non salta all'ottimo immediatamente.

Implicazioni pratiche In ambienti pericolosi (es. robotica, ambienti con penalità), SARSA è spesso preferito:

- SARSA tiene conto della **penalità di esplorazione** della politica ϵ-greedy che sta seguendo. Se l'esplorazione può portare a stati di costo molto elevato, SARSA apprenderà una politica che eviterà quelle regioni, anche se la politica ottima Q^* potrebbe teoricamente passare attraverso di esse con bassissima probabilità.
- Q-learning, puntando all'ottimo Q^*, non tiene conto dell'esplorazione e potrebbe apprendere un percorso che, pur essendo ottimo, è rischioso se l'agente esplora con la politica ϵ-greedy.

11.7 Approssimazione della Funzione di Valore (Value Function Approximation)

Nel caso di spazi di stato enormi (continuo o discreto ma molto grande), non è possibile memorizzare i Q-fattori in una tabella. Si usa la **Value Function Approximation** (VFA).

$$\tilde{V}(s; \theta) \quad \text{o} \quad \tilde{Q}(s, a; \theta).$$

Aggiornamento dei Parametri L'obiettivo è minimizzare l'errore tra il Q-fattore approssimato e la stima del Q-fattore ottenuta dal campionamento (il termine tra parentesi quadre nell'aggiornamento Q-learning/SARSA). Utilizzando il gradiente stocastico, si aggiornano i parametri θ in base al gradiente dell'errore (o *loss function*):

$$\theta_{t+1} = \theta_t + \alpha \left[\text{Target Value} - \tilde{Q}(s_t, a_t; \theta_t) \right] \cdot \nabla \tilde{Q}(s_t, a_t; \theta_t).$$

- **Target Value (Q-Learning)**: $r_{t+1} + \gamma \max_a \tilde{Q}(s_{t+1}, a; \theta_t)$.
- **Target Value (SARSA)**: $r_{t+1} + \gamma \tilde{Q}(s_{t+1}, a_{t+1}; \theta_t)$.

Questo è il fondamento del **Deep Reinforcement Learning** (es. Deep Q-Networks - DQN), dove la funzione di approssimazione $\tilde{Q}$ è una Rete Neurale Profonda.

12 Simulazione

Scrivere un paragrafo per fare recap dei problemi affrontati di simulazione.