

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**: $\mathbb{E}[(X - x)^2]$. Sviluppando il valore atteso, si ottiene

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

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**, $\mathbb{E}[|X - x|]$. 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

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

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**:

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

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

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

Questa formulazione mette in evidenza che, in generale, $\mathbb{E}[f(x, \tilde{\xi})] \neq f(x, \mathbb{E}[\tilde{\xi}])$, 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 $q = \mathbb{E}[D]$. 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:

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

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:

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

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

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

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

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**:

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

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

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

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

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

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 $j = 1, \dots, n$, e con d_j , $j = 1, \dots, n$ 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

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

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, \dots, 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 $PD \rightarrow PO$, 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 ,

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

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'innescò 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 $(A \text{ or } B) \text{ and } (\text{not}(A) \text{ or } C)$; definita rispetto alle variabili Booleane A , B e C , è soddisfatta se B e C sono entrambe vere. Al contrario, $(A \text{ or } B) \text{ and } (\text{not}(A) \text{ or } B) \text{ and } (\text{not}(B))$ 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

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

Creiamo un ulteriore job J_0 con parametri $r_0 = b$, $p_0 = 1$, $d_0 = b + 1$. 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:

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

Definiamo la funzione valore

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

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 $V_k(s)$, $k = 1, \dots, n$, 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:

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

Se il problema è della forma:

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

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 angolarizzazione), 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:

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

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

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

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:

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

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

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

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:

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

Il coordinatore regola il prezzo μ per portare i sottodecisi 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):

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

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.

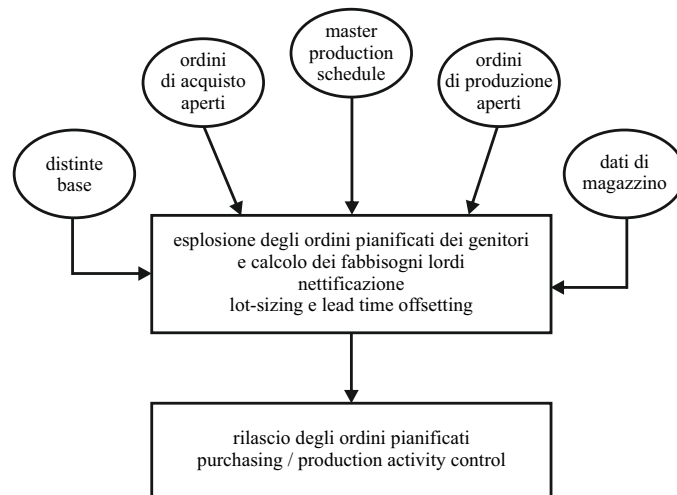
Negli anni Settanta sono stati introdotti i sistemi **MRP (Material Requirements Planning)**, basati su una **logica a capacità infinita**. Il vincolo di capacità produttiva viene rilassato, consentendo un **disaccoppiamento parziale tra item**, ma non tra domanda indipendente e dipendente.

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:

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

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

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

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

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

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:

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

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

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

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

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

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

4.12 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:

$$p_i T_i = d_i T_c \quad \Rightarrow \quad T_i = \frac{d_i}{p_i} T_c$$

Inoltre, il periodo di rotazione deve soddisfare il vincolo:

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

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

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

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

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

5 CAP 5 – Schedulazione nella produzione e nei servizi

Recap La **schedulazione** consiste nell'assegnare **risorse** a **job** nel tempo, rispettando **vincoli tecnologici** e di **capacità** e ottimizzando una **misura di prestazione**; rispetto alla **pianificazione della produzione** (es., **MRP**) opera a un livello di dettaglio superiore. I job sono descritti da **operazioni** con **precedenze**, **tempi di processo**, **tempi di rilascio** e **due date**; tipicamente si assume lavorazione su **una macchina per volta**, **una sola operazione per macchina** e **assenza di interruzioni**, pur con possibili violazioni (lot streaming, batch, pre-emption).

Le strutture di problema derivano da diverse **precedenze** e layout di macchine (**macchina singola**, **parallele** $P/Q/R$, **flow shop**, **job shop**, **open shop**), e le prestazioni si misurano tramite funzioni di penalità sui tempi di completamento (C_i , F_i , L_i , T_i , E_i , U_i), anche in forma **aggregata** (min-sum o min-max, con **pesi** w_i). Alcune misure risultano **equivalenti** fino a costanti (lateness totale e flow time totale) e L_{\max} implica ottimalità anche per T_{\max} , con $T_{\max} = \max\{L_{\max}, 0\}$. Le misure **regolari** (non decrescenti in C_i) permettono di ricavare il **timing** dal **sequenziamento** tramite **soluzioni attive**; con misure **non regolari** (es., somma pesata di earliness e tardiness) sequencing e timing non coincidono e la soluzione ottima non è necessariamente attiva o semiattiva.

La notazione di **Graham** $\alpha|\beta|\gamma$ classifica i problemi tramite layout, vincoli aggiuntivi e misura obiettivo. Esistono casi risolvibili in modo **polinomiale** (regole **EDD**, **WSPT**, algoritmo di **Johnson**), ma in generale i problemi sono **NP-hard** e si ricorre a **euristiche** e **metaeuristiche**, includendo regole avanzate come **ATC**. La formulazione tramite **grafo disgiuntivo** rappresenta precedenze (archi congiuntivi) e vincoli di capacità (archi disgiuntivi), e il **cammino critico** determina il **makespan**; da tale grafo si può derivare un **MILP** per $J//C_{\max}$, utile come base per strategie *destroy and repair* pur non essendo trattabile in pratica. Nella ricerca locale, perturbare archi disgiuntivi sul **cammino critico** evita cicli ed è l'unica perturbazione utile per $J//C_{\max}$.

La procedura **shifting bottleneck** decompone $J//C_{\max}$ in problemi su macchina singola del tipo $1/r_i/L_{\max}$, usando **teste** h_{ib} (rilasci r_{ib}) e **code** t_{ib} per costruire **due date locali** $d_{ib} = K - t_{ib}$; il collo di bottiglia è la macchina con L_{\max} peggiore e viene congelata iterativamente fino a schedulare tutte le macchine, con possibili raffinamenti ed estensioni a $J/r_i/L_{\max}$.

5.1 Modelli classici di machine scheduling

Risolvere un problema di **schedulazione** richiede di assegnare **risorse** a **job** da eseguire nel tempo, rispettando **vincoli tecnologici** e di **capacità**, in modo da ottimizzare una **misura di prestazione**.

Un problema di scheduling si pone a un livello di dettaglio superiore rispetto a un problema di **pianificazione della produzione** (ad esempio a livello **MRP**). Esiste una grande varietà di problemi di scheduling, ma si inizia con la definizione di un **problema minimale**.

Caratteristiche dei job I job sono caratterizzati da:

- Un **set di operazioni** da eseguire, legate da **vincoli di precedenza**;
- **Tempi di lavorazione** per l'esecuzione su ogni macchina;

- **Date di rilascio e date di consegna.**

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

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

Dall'esempio si ricava che una stessa istanza può ammettere più soluzioni ammissibili, caratterizzate da diverse sequenze sulle macchine.

5.2 Tipi di flusso

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

Il **flusso dei materiali** caratterizza diverse strutture in termini di macchine:

- **macchina singola**;
- **macchine parallele** (identiche: $p_{im} = p_i$, correlate: $p_{im} = p_{ism}$, scorrelate);
- **flow shop**;
- **job shop**;
- **open shop**.

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

5.3 Misure di prestazione

Si indica con C_i il **tempo di completamento** del job $i \in [n]$, ovvero il tempo di completamento della sua **ultima operazione**.

Funzioni di penalità Un tipico modo per costruire misure di prestazione è associare ad ogni job J_i un **termine di penalità** $\gamma_i(C_i)$. I termini tipicamente utilizzati sono:

- **tempo di completamento**: $\gamma_i(C_i) = C_i$;
- **flow time**: $\gamma_i(C_i) = F_i = C_i - r_i$;
- **lateness**: $\gamma_i(C_i) = L_i = C_i - d_i$;
- **tardiness**: $\gamma_i(C_i) = T_i = \max\{C_i - d_i, 0\}$;
- **earliness**: $\gamma_i(C_i) = E_i = \max\{d_i - C_i, 0\}$;
- **indicatore di ritardo**: $\gamma_i(C_i) = U_i = \begin{cases} 1 & \text{se } C_i > d_i, \\ 0 & \text{se } C_i \leq d_i. \end{cases}$

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

Sulla base di questi “**mattoncini**” si possono costruire misure di prestazione **aggregate**, ottenendo funzioni del tipo:

$$\min \sum_{i=1}^n \gamma_i(C_i), \quad \min \max_{i=1, \dots, n} \gamma_i(C_i).$$

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

- **flow time totale**: $\sum_i F_i$;
- **flow time totale pesato**: $\sum_i w_i F_i$;
- **massima lateness**: $L_{\max} = \max_i L_i$;
- **tardiness totale pesata**: $\sum_i w_i T_i$;
- **makespan**: $C_{\max} = \max_i C_i$;
- **numero di job in ritardo**: $\sum_i U_i$.

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

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

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

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

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

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

In questo caso esiste una regione in cui la funzione **decresce** al crescere del tempo di completamento, e **sequencing e timing non coincidono** e la soluzione ottima non è necessariamente attiva o semiattiva.

5.4 Notazione di Graham

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

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

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

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

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

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

5.5 Algoritmi di soluzione

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

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

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

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

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

dove w_i è il peso del job J_i e p_{ij} è il suo tempo di processo sulla macchina M_j ; la notazione $[x]^+$ è equivalente a $\max\{0, x\}$; W_{iq} è una stima del tempo di attesa di J_i sulla macchina M_q ; m_i è il numero di macchine che il job deve ancora visitare; k è un parametro da selezionare; \bar{p} è il tempo di processo medio dei job in coda.

Interpretazione del termine di slack L'espressione

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

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

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

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

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

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

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

I principali problemi sono:

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

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

5.6 Grafi disgiuntivi

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

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

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

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

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

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

5.8 Procedura Shifting Bottleneck

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

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

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

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

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

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

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

6 CAP 6 – Pricing

7 CAP 7 – Il principio della programmazione dinamica

8 CAP 8 – Implementazione della programmazione dinamica

8.1 Allocazione discreta di risorse: il problema dello zaino

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

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

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

Il problema può essere formulato come un problema di **programmazione lineare binaria pura**:

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

Si assume un'allocazione discreta del budget, poiché la selezione di un'attività è una decisione **tutto-o-niente**.

Motivazione dell'approccio DP Il problema può essere risolto in modo efficiente con algoritmi **branch-and-cut** per la programmazione lineare intera, ma qui si adotta un approccio basato sul principio della **programmazione dinamica**.

8.2 Riformulazione sequenziale del problema

Indice temporale fittizio e stadi decisionali Il problema non è intrinsecamente dinamico, ma può essere riformulato come un problema di **allocazione sequenziale di risorse** introducendo un indice discreto fittizio k , associato agli oggetti.

Per ogni stadio $k = 1, \dots, n$, si decide se includere o meno l'oggetto k nel sottoinsieme.

Stato e dipendenza dalle decisioni passate Allo stadio $k = 1$ si dispone del budget B e si deve scegliere se includere il primo oggetto, affrontando un compromesso tra **ricompensa immediata** e **ricompense future**. Allo stadio $k = n$ il problema è banale, poiché resta da valutare solo l'ultimo oggetto dato il budget residuo.

La selezione degli oggetti successivi dipende dalle decisioni passate soltanto attraverso il **budget residuo**. La variabile di stato naturale allo stadio k è il budget disponibile s_k prima di selezionare l'oggetto k , mentre la variabile decisionale è la variabile binaria x_k .

Equazione di transizione dello stato Poiché non esiste l'oggetto $k = 0$, si usa la transizione:

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

con condizione iniziale

$$s_1 = B.$$

Assumendo che i pesi w_k siano **interi**, anche la variabile di stato assume valori **interi**.

8.3 Funzione valore e ricorsione DP

Definizione della funzione valore Si definisce la funzione valore:

$$V_k(s) := \text{profitto della selezione ottimale degli oggetti } \{k, k+1, \dots, n\} \text{ con budget residuo } s.$$

Nel caso discreto, è possibile **tabulare** tutte le funzioni $V_k(s)$ per $k = 1, \dots, n$ e $s \in \{0, \dots, B\}$.

Equazione di Bellman La ricorsione DP è:

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

Interpretazione della ricorsione Allo stadio k , si considera l'oggetto k . Se $s < w_k$, l'oggetto non entra nel budget residuo e quindi la funzione valore resta **invariata**. Se $s \geq w_k$, si confrontano due alternative: includere l'oggetto (ottenendo v_k e aggiornando lo stato a $s - w_k$) oppure escluderlo (lasciando lo stato a s). Poiché la decisione è **binaria**, l'ottimizzazione a singolo stadio è immediata.

Condizione terminale Per l'ultimo oggetto $k = n$:

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

8.4 Implementazione MATLAB del problema dello zaino

Obiettivo della funzione La funzione `DPKnapsack` ha lo scopo di calcolare la soluzione ottima del problema dello **zaino** dato un vettore di **valori**, un vettore di **pesi** e una **capacità**. L'output è la selezione ottima come vettore **binario** e il **valore totale** ottenuto.

Idea di memorizzazione: tabelle di valore e decisione La funzione valore viene memorizzata in una tabella `valueTable`, indicizzata per **stadio** (oggetto) e per **stato** (budget residuo). Poiché MATLAB indicizza a partire da 1, lo stato s corrisponde alla riga $s + 1$.

In parallelo, la tabella `decisionTable` memorizza, per ogni coppia **stato**–**stadio**, la **decisione ottima** (includere o non includere).

Ricorsione backward e ricostruzione forward La tabulazione implementa una ricorsione **backward** sugli stadi, calcolando le funzioni valore a ritroso. Successivamente, la soluzione ottima viene ricostruita procedendo **forward** sugli oggetti, usando la tabella delle decisioni e aggiornando lo stato (budget residuo) coerentemente.

Complessità computazionale Assumendo dati interi, la complessità è $O(nB)$, quindi **pseudo-polinomiale**. Quando B è grande, occorre costruire una tabella molto grande e l'algoritmo può diventare poco efficiente.

8.5 Allocazione continua del budget

Formulazione del problema Si considera una versione **continua** del problema di allocazione delle risorse: un budget B deve essere allocato tra n attività, con decisioni continue $x_k \geq 0$, $k = 1, \dots, n$. Il contributo di profitto dell'attività k dipende dall'allocazione tramite una funzione $f_k(\cdot)$ **crescente e concava**:

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

Uso completo del budget Poiché le funzioni di profitto sono **strettamente crescenti**, si può assumere che il budget sia interamente utilizzato, quindi il vincolo di budget è soddisfatto come **uguaglianza** nella soluzione ottima.

Soluzione interna e Lagrangiana Assumendo una soluzione interna $x_k^* > 0$, il problema diventa un problema non lineare con un solo vincolo di uguaglianza. Introducendo il moltiplicatore di Lagrange λ , la lagrangiana è:

$$L(x, \lambda) = \sum_{k=1}^n \sqrt{x_k} + \lambda \left(\sum_{k=1}^n x_k - B \right).$$

Le condizioni del primo ordine sono:

$$\frac{1}{2\sqrt{x_k}} + \lambda = 0, \quad k = 1, \dots, n,$$

$$\sum_{k=1}^n x_k - B = 0,$$

che implicano un'allocazione **uniforme**:

$$x_k^* = \frac{B}{n}, \quad k = 1, \dots, n.$$

In questo caso, la funzione obiettivo è **concava** e le condizioni di ottimalità sono **necessarie e sufficienti**.

8.6 Allocazione continua del budget: formulazione DP

Riformulazione DP Il problema può essere riformulato in un quadro di **programmazione dinamica** introducendo un indice discreto fittizio $k = 1, \dots, n$ associato alle attività. Sia $V_k(s)$ il profitto ottimale ottenibile allocando un budget residuo s alle attività $\{k, k+1, \dots, n\}$.

Transizione dello stato La transizione dello stato è:

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

con condizione iniziale $s_1 = B$.

Equazioni di ottimalità Le funzioni valore soddisfano:

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

con condizione terminale:

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

Il vincolo $s_k \geq 0$ è garantito imponendo il vincolo sulla decisione $0 \leq x_k \leq s_k$, poiché la transizione è deterministica.

Funzioni valore in spazio infinito-dimensionale Nel caso continuo, la funzione valore $V_k(s)$ è un oggetto in uno spazio funzionale **infinito-dimensionale**. Poiché la valutazione è possibile solo su un insieme finito di stati, occorre usare **approssimazione** o **interpolazione** per stimare i valori fuori dalla griglia.

8.7 Intermezzo: interpolazione con spline cubiche in MATLAB

Limiti dell'interpolazione polinomiale È noto che l'interpolazione polinomiale può soffrire di **oscillazioni inaccettabili**. Una funzione **concava** e **monotona crescente** può essere approssimata da una funzione **non monotona**, con effetti negativi sulle procedure di ottimizzazione.

Spline cubiche come alternativa Un'alternativa standard consiste nell'uso di funzioni polinomiali a tratti di ordine più basso, in cui ogni tratto è associato a un sottointervallo della griglia. Una scelta comune sono le **spline cubiche**, ottenute in MATLAB tramite **spline** (costruzione) e **ppval** (valutazione in punti arbitrari, anche fuori griglia).

Accuratezza, griglia e questioni aperte L'errore può essere significativo con griglie grossolane e migliora con griglie più fitte. Restano aperti temi come: la garanzia che una funzione monotona venga approssimata da una spline monotona, la scelta dei nodi per un buon compromesso tra costo computazionale ed errore, e la generalizzazione a dimensioni superiori.

8.8 Risoluzione numerica del problema continuo tramite DP

Idea della discretizzazione e interpolazione Si usa una **spline cubica** per approssimare la funzione valore del problema continuo. Si imposta una griglia uniforme su $[0, B]$, replicata per ciascuno stadio. La griglia contiene $m + 1$ valori di stato, con passo $\delta s = B/m$, cioè stati del tipo $j \delta s$, $j = 0, 1, \dots, m$.

Risoluzione dei sottoproblemi e policy implicita Per ogni punto della griglia si risolve un sottoproblema del tipo:

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

tramite ottimizzazione numerica. I valori fuori dalla griglia sono stimati via spline cubiche. La condizione al bordo su $V_n(\cdot)$ è banale e si arresta il calcolo a $V_2(\cdot)$ quando si vuole risolvere il problema per un budget specifico B . La politica ottima $x_t^* = \mu_t^*(s_t)$ non è memorizzata in una tabella esplicita, ma è **implicita** nella sequenza delle funzioni valore.

8.9 Implementazione MATLAB: idea generale

Scopo di findPolicy La funzione **findPolicy** serve a costruire, stadio per stadio, un'**approssimazione** delle funzioni valore tramite **spline cubiche**, usando una griglia uniforme del budget. Riceve il budget totale, una lista di funzioni di profitto (una per attività) e il numero di punti della griglia. Restituisce una lista di spline che rappresentano le funzioni valore approssimate; la prima funzione non viene usata direttamente quando l'interesse è risolvere il problema per un budget iniziale specifico.

Scopo di applyPolicy La funzione **applyPolicy** applica la politica ottima **in avanti nel tempo** utilizzando le funzioni valore approssimate: a ogni stadio sceglie l'allocazione che massimizza la somma tra contributo immediato e valore futuro, aggiorna il budget residuo e costruisce il vettore delle allocazioni ottime insieme al valore complessivo ottenuto.

8.10 Controllo stocastico delle scorte

Impostazione del problema Si considera una variazione **stocastica** del problema di lot-sizing, assumendo una **domanda aleatoria discreta**. In questo caso è naturale adottare una rappresentazione **tabellare** della funzione valore.

Dinamica dello stato con stockout (lost sales) Occorre specificare la dinamica dello stato in caso di stockout. Qui si assumono **vendite perse**:

$$I_{t+1} = \max\{0, I_t + x_t - d_{t+1}\}$$

dove $(d_t)_{t=1, \dots, T}$ è una sequenza di variabili i.i.d., e x_t è la quantità ordinata al tempo t e consegnata immediatamente.

Sequenza degli eventi Al tempo t si osserva l'inventario a mano I_t . Poi si decide l'ordine x_t , che viene ricevuto immediatamente e porta la disponibilità a $I_t + x_t$. Durante l'intervallo $t + 1$ si osserva la domanda aleatoria d_{t+1} e si aggiorna l'inventario secondo la dinamica sopra.

Spazio degli stati e azioni ammissibili Per tabulare la funzione valore occorre fissare un limite superiore allo stato. Si assume un vincolo $I_t \leq I_{\max}$, che implica:

$$X(I_t) = \{0, 1, \dots, I_{\max} - I_t\}.$$

Costo immediato e stato terminale Il costo immediato include un costo lineare d'ordine $c x_t$ e una penalità quadratica sull'inventario contabile dopo la domanda del periodo successivo:

$$\phi(I_t + x_t - d_{t+1})^2.$$

L'inventario fisico non può essere negativo ed è determinato dalla dinamica con $\max\{0, \cdot\}$; un inventario contabile negativo rappresenta domanda non soddisfatta. Per semplicità, il costo terminale è nullo:

$$F_T(I_T) = 0.$$

Ricorsione DP con costo immediato stocastico Il costo immediato dipende dalla realizzazione del fattore aleatorio nel periodo successivo alla decisione; quindi nella ricorsione compare un termine **stocastico**. La ricorsione è:

$$V_t(I_t) = \min_{x_t \in X(I_t)} \mathbb{E}_{d_{t+1}} \left[c x_t + \phi(I_t + x_t - d_{t+1})^2 + V_{t+1}(\max\{0, I_t + x_t - d_{t+1}\}) \right]$$

per $t = 0, 1, \dots, T - 1$ e $I_t \in \{0, 1, 2, \dots, I_{\max}\}$.

Modellare l'incertezza Poiché la domanda è discreta e i.i.d., è sufficiente una **funzione di massa di probabilità**, cioè un vettore π_k per i possibili valori $k = 0, 1, \dots, d_{\max}$. Occorre inoltre specificare lo stato iniziale I_0 e l'orizzonte T .

8.11 Lot-sizing stocastico: idea delle procedure MATLAB

Scopo di MakePolicy La funzione `MakePolicy` serve ad apprendere le funzioni valore **a ritroso (backward)** e la **politica ottima** in forma tabellare, dato un limite massimo di inventario, una distribuzione discreta della domanda e i parametri economici. L'output è una tabella dei valori (funzione valore per ogni stato e istante) e una tabella delle azioni (decisione ottima d'ordine per ogni stato e istante).

Scopo di SimulatePolicy La funzione `SimulatePolicy` serve a **simulare** l'applicazione della politica ottima su molti scenari di domanda generati casualmente, per ottenere una stima statistica del costo totale e confrontarla con il valore previsto dalla funzione valore.

8.12 Sfruttare la struttura: cammini minimi per il lot-sizing deterministico

Motivazione Nel problema toy di lot-sizing è semplice memorizzare funzioni valore e politica in una tabella, ma ciò è **inefficiente** quando i valori possibili di domanda sono molti, e diventa impossibile con spazio degli stati **continuo**. Talvolta è possibile semplificare drasticamente il problema sfruttando la **struttura**.

Lot-sizing deterministico con costi fissi e holding Si consideri una versione deterministica con soli **costi fissi d'ordine** ϕ e **costi di giacenza** h , con inventario iniziale e finale pari a zero, e domanda non nulla nel primo periodo. In linea di principio, una ricorsione DP è:

$$V_t(I_t) = \min_{x_t \geq d_{t+1} - I_t} \{ \phi \delta(x_t) + h(I_t + x_t - d_{t+1}) + V_{t+1}(I_{t+1}) \}, \quad t = 0, \dots, T-1$$

con condizione:

$$V_T(I_T) \equiv 0.$$

Il vincolo su x_t garantisce che la domanda sia sempre soddisfatta e che lo stato non diventi negativo.

Teorema: proprietà di Wagner–Whitin Per il lot-sizing deterministico non capacitated con **costi fissi** e **costi lineari di inventario**, esiste una soluzione ottima tale che:

$$I_t x_t = 0, \quad t = 0, 1, \dots, T-1.$$

Il messaggio è che non è mai ottimale ordinare quando l'inventario è positivo: si ordina solo se l'inventario è **nullo**.

Rete di flusso e interpretazione Considerando una rappresentazione come rete, deve valere il bilancio globale:

$$\sum_{t=0}^{T-1} x_t = \sum_{t=1}^T d_t.$$

Si introduce un nodo fittizio 0 il cui inflow rappresenta l'ammontare totale ordinato. I nodi $t = 1, \dots, T$ rappresentano gli istanti associati alla fine di ciascun intervallo, cioè la domanda si soddisfa alla fine dell'intervallo.

Il bilancio al nodo t corrisponde alla transizione:

$$I_t = I_{t-1} + x_{t-1} - d_t.$$

Se, contrariamente al teorema, $I_{t-1} > 0$ e $x_{t-1} > 0$, allora reindirizzando l'inflow orizzontale I_{t-1} lungo l'arco d'ordine associato a x_{t-1} si può migliorare il costo totale.

Conseguenza pratica: scelte d'ordine candidate All'istante t basta considerare solo:

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

Riformulazione come cammino minimo Con questa proprietà, il problema a singolo item si riformula come un **cammino minimo** su una rete piccola: dal nodo iniziale 0 al nodo terminale, scegliendo archi che rappresentano quanti periodi vengono coperti dal prossimo ordine. Il costo di ciascun arco incorpora il **costo fisso d'ordine** e il **costo di giacenza** risultante. Poiché la rete ha un numero limitato di nodi, questa formulazione porta a un algoritmo molto efficiente di complessità **polinomiale**.

8.13 Lot-sizing stocastico: politiche S e (s, S)

Backlog e penalità convessa La proprietà di Wagner–Whitin non vale nel caso stocastico, ma in alcuni casi esistono risultati strutturali. Si assume che non vi siano **vendite perse** e che sia consentito backlog, con costo totale che include:

$$q(s) = h \max\{0, s\} + b \max\{0, -s\},$$

dove s può essere positivo (inventario) o negativo (backlog), h è il costo di giacenza e $b > h$ è il costo di backlog. La funzione $q(\cdot)$ è **convessa** e tende a $+\infty$ quando $s \rightarrow \pm\infty$.

Costo variabile e obiettivo atteso Trascurando i costi fissi e includendo un costo variabile lineare unitario c , l'obiettivo è minimizzare:

$$\mathbb{E}_0 \left[\sum_{t=0}^{T-1} (c x_t + q(I_t + x_t - d_{t+1})) \right].$$

Ricorsione DP e funzione $H(\cdot)$ La ricorsione DP è:

$$V_t(I_t) = \min_{x_t \geq 0} \{c x_t + H(I_t + x_t) + \mathbb{E}[V_{t+1}(I_t + x_t - d_{t+1})]\},$$

dove:

$$H(y_t) := \mathbb{E}[q(y_t - d_{t+1})] = h \mathbb{E}[\max\{0, y_t - d_{t+1}\}] + b \mathbb{E}[\max\{0, d_{t+1} - y_t\}].$$

Si introduce y_t come inventario disponibile dopo l'ordine (lead time nullo):

$$y_t := I_t + x_t,$$

con condizione terminale:

$$V_T(I_T) = 0.$$

Si assume distribuzione della domanda costante nel tempo.

Riscrittura tramite $G_t(\cdot)$ La ricorsione si riscrive come:

$$V_t(I_t) = \min_{y_t \geq I_t} G_t(y_t) - c I_t,$$

dove:

$$G_t(y_t) = c y_t + H(y_t) + \mathbb{E}[V_{t+1}(y_t - d_{t+1})].$$

Convessità e livello obiettivo S_t Si assume (senza prova) che $V_t(\cdot)$ e $G_t(\cdot)$ siano **convesse** per ogni t , e che $G_t(\cdot) \rightarrow +\infty$ quando $y \rightarrow \pm\infty$. Ne segue che $G_t(\cdot)$ ha un minimizzatore non vincolato finito:

$$S_t = \arg \min_{y_t \in \mathbb{R}} G_t(y_t).$$

Politica base-stock (order-up-to) La politica ottima è una politica **base-stock**:

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

I valori S_t sono livelli obiettivo: si ordina quanto serve per raggiungere il target ottimo. In orizzonte finito, il problema è determinare la sequenza ottima di livelli S_t .

Politiche (s, S) con costi fissi Se si introducono costi fissi, si perde convessità. Tuttavia una proprietà correlata (K-convessità) porta a una politica ottima:

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

dipendente da due sequenze s_t e S_t , con $s_t \leq S_t$. In ambiente stazionario è ottima una politica stazionaria (s, S) . Si ordina solo quando l'inventario è sotto **small** s , riportandolo a **big** S ; $S - s$ è una quantità minima d'ordine che aiuta a controllare i costi fissi.

8.14 Le maledizioni della programmazione dinamica

Limiti della DP La DP è potente e flessibile, ma presenta limitazioni importanti.

Maledizione della dimensionalità dello stato Serve la funzione valore per ogni elemento dello spazio degli stati. Se lo spazio è finito e piccolo, si possono usare tabelle; per spazi enormi ciò non è fattibile.

Maledizione dell'ottimizzazione La DP decompone un problema multistadio in sottoproblemi a singolo stadio, ma anche questi sottoproblemi possono essere difficili da risolvere.

Maledizione dell'aspettativa Se i fattori di rischio ξ_t sono continui, l'aspettativa richiede integrali multidimensionali difficili; serve quindi una discretizzazione.

Maledizione della modellazione Il sistema può essere troppo complesso per avere un modello esplicito delle transizioni di stato. In DP il problema è più complicato perché le transizioni sono almeno in parte influenzate dalle decisioni di controllo.

9 CAP 9 – Modelli della programmazione dinamica

10 CAP 9 – Modeling for Dynamic Programming

10.1 Frontespizio e riferimenti

Titolo e autore Business Analytics (2020/21). **Modeling for dynamic programming**. Prof. Paolo Brandimarte, Dipartimento di Scienze Matematiche, Politecnico di Torino.

Riferimenti Le slide sono tratte dal libro: **P. Brandimarte**, *From Shortest Paths to Reinforcement Learning: A MATLAB-Based Introduction to Dynamic Programming*, Springer, 2021. Ulteriori riferimenti: P. Brandimarte, *An Introduction to Financial Markets: A Quantitative Approach*, Wiley, 2018; J.Y. Campbell, L.M. Viceira, *Strategic Asset Allocation*, Oxford University Press, 2002; W.B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (2nd ed.), Wiley, 2011; G.J. van Ryzin, K.T. Talluri, *An Introduction to Revenue Management*, INFORMS Tutorials in OR, 2005.

10.2 Principio di modellazione nella programmazione dinamica

Idea generale della programmazione dinamica Il principio della **programmazione dinamica (DP)** è un concetto **flessibile** per la decomposizione di un **problema decisionale multistadio** in una sequenza di **problemi a singolo stadio**, bilanciando il **contributo immediato** e i **contributi attesi** delle decisioni future.

Funzione valore e ricorsione DP Il principio si fonda sulla **funzione valore**, definita dalla ricorsione di DP:

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

Forme alternative della ricorsione DP L'Eq. (1) è solo **una possibile forma** della ricorsione di DP. Si possono adottare forme più specifiche, ad esempio quando i **fattori di rischio** sono **variabili aleatorie discrete** e l'aspettazione si riduce a una **somma**.

Scambio tra aspettazione e ottimizzazione In alcuni casi si adotta una riformulazione più radicale, in cui si **scambiano** gli operatori di **aspettazione** e **ottimizzazione**. Ciò può dipendere dalla **struttura informativa del problema** oppure essere il risultato di una manipolazione volta a evitare la soluzione di un **sottoproblema stocastico difficile**.

Q-learning e stati post-decisione Un caso ben noto è il **Q-learning**, una forma di **reinforcement learning** in cui $V(s)$ è sostituita da **Q-factors** $Q(s, x)$, che rappresentano il valore delle **coppie stato-azione**. Più in generale, lo scambio tra ottimizzazione e aspettazione può essere ottenuto introducendo il concetto di **stato post-decisione**.

Scelta della rappresentazione dello stato Trovare una descrizione adeguata dello **stato del sistema** può richiedere un'attenta modellazione: può essere necessaria una **ridefinizione dello spazio degli stati** per eliminare **dipendenze dal percorso** e ottenere un modello **markoviano aumentato ed equivalente**.

Natura degli stati e delle decisioni Sebbene sia naturale pensare a stati e decisioni come **scalari** o **vettori**, essi possono consistere in oggetti diversi, ad esempio **insiemi**.

Ruolo delle capacità di modellazione Le **capacità di modellazione** procedono di pari passo con la **conoscenza algoritmica** nell'affrontare un problema decisionale tramite DP; tali capacità si affinano solo con l'esperienza su una **vasta gamma di problemi**.

10.3 Processi decisionali di Markov finiti

Definizione di MDP Il termine **processo decisionale di Markov (MDP)** è riservato a problemi con **spazi di stato** e **spazi delle azioni discreti**. Il termine **azione** è spesso adottato per riferirsi alle decisioni di controllo. Stati e azioni discreti possono essere **enumerati** e associati a numeri interi.

MDP finiti e notazione degli stati Si considerano **MDP finiti**. Anche se gli stati possono corrispondere a vettori in uno spazio multidimensionale, si usa una notazione del tipo

$$i, j \in S = \{1, \dots, N\},$$

dove N è la **dimensione dello spazio degli stati**.

Azioni ammissibili Si usa a o a_t per indicare le azioni; $A_t(i)$ o $A(i)$ denota l'insieme delle **azioni ammissibili** nello stato i al tempo t . L'insieme di tutte le azioni possibili è A .

Dinamica markoviana controllata Il sistema sottostante è una **catena di Markov finita a tempo discreto**. Nei MDP le **probabilità di transizione** dipendono dall'azione selezionata:

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

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

10.4 Esempio: controllo stocastico di inventario

Sintesi dell'esempio Dall'esempio si evince come, con **spazio di stato finito** e **azioni ammissibili dipendenti dallo stato**, le **matrici di transizione** dipendano dall'azione e possano essere rappresentate in forma tabellare.

10.5 Ricorsioni DP per MDP

Ricorsione a orizzonte finito Nel contesto MDP, la funzione valore al tempo t è un vettore finito-dimensionale con componenti $V_t(i)$ e soddisfa:

$$V_t(i) = \text{opt}_{a \in A(i)} \left\{ f_t(i, a) + \gamma \sum_{j \in S} \pi_{t+1}(i, a, j) V_{t+1}(j) \right\}, \quad i \in S. \quad (2)$$

Caso a orizzonte infinito scontato Nel caso a **orizzonte infinito scontato**:

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

Contributo immediato stocastico Se il contributo immediato è stocastico e dipende dallo stato successivo, lo si denota $h(i, a, j)$ e si riscrive:

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

Relazione tra contributi In principio,

$$f(i, a) = \sum_{j \in S} \pi(i, a, j) h(i, a, j),$$

ma ciò può essere **impraticabile** quando N è finito ma enorme oppure quando le **probabilità di transizione non sono note**.

10.6 Esempio: arresto ottimo ricorrente

Sintesi dell'esempio Dall'esempio si evince il **trade-off** tra **ricompensa immediata** e **attesa** di opportunità migliori, e come la struttura della catena consenta una formulazione DP esplicita sui valori $V(k)$.

10.7 Valutazione delle politiche e Q-factors

Policy evaluation e policy iteration Una politica stazionaria ammissibile μ associa a ciascuno stato i un'azione $a = \mu(i) \in A(i)$. La funzione valore associata a μ si ottiene risolvendo un sistema lineare:

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

Questa relazione è fondamentale nei metodi basati su **policy iteration**: si valuta una politica candidata e poi si tenta di migliorarla.

Definizione di Q-factor per una politica Il **Q-factor** associato a μ è la mappa $S \times A \rightarrow \mathbb{R}$:

$$Q^\mu(i, a) := f(i, a) + \gamma \sum_{j \in S} \pi(i, a, j) V^\mu(j). \quad (6)$$

L'idea è applicare a nello stato i e poi seguire la politica μ .

Q-factors ottimi e relazione con V Sostituendo la funzione valore ottima in (6) si ottengono i **Q-factors ottimi**. Vale:

$$V(j) \equiv \text{opt}_{a \in A(j)} Q(j, a), \quad j \in S.$$

Ricorsione DP in termini di Q-factors La ricorsione di DP può essere riscritta come:

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

Vantaggi e svantaggi Lo **svantaggio** è che si passa da $V(i)$ a funzioni **stato-azione** $Q(i, a)$, aggravando potenzialmente la **curse of dimensionality**. Il **vantaggio** è lo **scambio** tra **aspettazione** e **ottimizzazione**, utile anche perché i Q-factors possono essere **appresi tramite campionamento statistico** (DP **model-free**) e, se necessario, rappresentati con architetture di approssimazione dalla **regressione lineare** alle **reti neurali profonde**.

10.8 Diverse forme di DP stocastica

Ricorsione DP di base Si consideri nuovamente la ricorsione di DP:

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

Ipotesi sottostanti Si osserva lo **stato** s_t all'istante t ; si prende una **decisione ammissibile** $x_t \in X(s_t)$; si osserva un **contributo immediato** $f_t(s_t, x_t)$; si passa a un nuovo stato s_{t+1} che dipende dai **fattori di rischio realizzati** ξ_{t+1} , secondo una distribuzione che può dipendere da s_t e x_t .

Difficoltà e motivazione dello scambio Per determinare $V_t(\cdot)$ a partire da $V_{t+1}(\cdot)$ si dovrebbe risolvere un problema di **ottimizzazione stocastica** che può includere un'**aspettazione** impegnativa; nei MDP i **Q-factors** consentono di scambiare **ottimizzazione** e **aspettazione**, e talvolta lo scambio è richiesto dalla **struttura informativa**.

10.9 Esempio: lot-sizing con lookahead limitato

Sintesi dell'esempio Dall'esempio si evince che, se la domanda del periodo successivo è **nota** al tempo t , la ricorsione naturale può assumere una forma in cui l'**ottimizzazione interna** è **deterministica** e l'**aspettazione** è **esterna**.

10.10 Variabili di stato post-decisione

Ricorsione in forma scambiata Una forma alternativa è:

$$V_t(s_t) = \mathbb{E}_t[\text{opt}_{x_t \in X(s_t)} \{f_t(x_t, s_t) + \gamma V_{t+1}(s_{t+1})\}]. \quad (9)$$

Vantaggi potenziali Il problema di ottimizzazione interno è **deterministico** e l'**aspettazione** esterna può essere stimata tramite **campionamento statistico**.

Introduzione dello stato post-decisione Si introduce uno stato intermedio osservato dopo la decisione x_t ma prima della realizzazione di ξ_{t+1} , detto **stato post-decisione** s_t^x . Si scinde la transizione in due passi:

$$s_t^x = g_t^{(1)}(s_t, x_t), \quad s_{t+1} = g_{t+1}^{(2)}(s_t^x, \xi_{t+1}). \quad (10-11)$$

Relazione tra funzioni valore Si definisce il valore del post-decision state $V_t^x(s_t^x)$ e vale:

$$V_t^x(s_t^x) = \mathbb{E}[V_{t+1}(s_{t+1}) \mid s_t^x]. \quad (12)$$

Quindi la ricorsione standard può essere riscritta come ottimizzazione deterministica:

$$V_t(s_t) = \text{opt}_{x_t \in X(s_t)} \{f_t(s_t, x_t) + \gamma V_t^x(s_t^x)\}. \quad (13)$$

Passo all'indietro e nuovo scambio Scrivendo la relazione un passo indietro e sostituendo (13), si ottiene una ricorsione che presenta nuovamente lo **scambio** tra **aspettazione** e **ottimizzazione**:

$$V_{t-1}^x(s_{t-1}^x) = \mathbb{E} \left[\text{opt}_{x_t \in X(s_t)} (f_t(s_t, x_t) + \gamma V_t^x(s_t^x)) \mid s_{t-1}^x \right]. \quad (14)$$

Connessione con i Q-factors La coppia **stato-azione** (i, a) può essere interpretata come **stato post-decisione** prima di osservare la transizione casuale al nuovo stato; la formulazione in termini di **Q-factors** si inserisce naturalmente in questo quadro.

10.11 Aumento dello stato nella gestione delle scorte

Inventario on-hand vs inventario disponibile L'**inventario on-hand** appare naturale come variabile di stato, ma le decisioni d'ordine dovrebbero basarsi sull'**inventario disponibile**, che include **on-order** e **backlog**.

Limite dell'equazione di stato standard L'equazione

$$I_{t+1} = I_t + x_t - d_{t+1}$$

assume disponibilità immediata di quanto ordinato al tempo t .

Lead time e variabili di pipeline Se il **lead time** è un intero $LT \geq 1$, si introducono variabili $z_{t,\tau}$ (pipeline), con $\tau = 0, 1, \dots, LT - 1$. Vale:

$$I_{t+1} = I_t + z_{t,0} - d_{t+1}$$

(trascurando l'incertezza) e la decisione x_t entra come:

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

Per $\tau < LT - 1$, le transizioni sono uno shift temporale:

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

Items deperibili e stato per età Per prodotti **deperibili** si introduce un array $I_{t,\tau}$ che descrive l'inventario per età; le transizioni dipendono dalla politica di issuing **FIFO** o **LIFO**.

10.12 Revenue management

Definizione e obiettivo Il **revenue management** consiste in modelli e tecniche per massimizzare il **ricavo** ottenuto vendendo **risorse deperibili**; l'idea nasce (come **yield management**) nell'industria aerea.

Approcci quantity-based e price-based Esistono due approcci: **quantity-based** (si limita la disponibilità della risorsa per massimizzare il ricavo) e **price-based** (si aggiustano dinamicamente i prezzi, **dynamic pricing**). Si considerano modelli DP quantity-based assumendo costo marginale nullo o trascurabile, così che massimizzare il profitto equivale a massimizzare il ricavo.

Classi tariffarie e prezzi Si considerino C unità di risorsa da allocare a n classi $j = 1, \dots, n$, vendute a prezzi p_j con:

$$p_1 > p_2 > \dots > p_n.$$

I posti sono fisicamente **identici**, ma differenziati tramite **ancillaries** (cancellazione, vincoli, pasti, ecc.).

Caratteristiche essenziali La domanda D_j è casuale e può realizzarsi con schemi diversi. Due aspetti sono essenziali: **comportamento dei clienti** (segmentazione perfetta vs preferenze da modellare con un **choice model**) e **timing della domanda** (sequentialità auspicabile; caso peggiore quando arrivano prima i clienti low-budget).

Modelli statici e dinamici; protection levels e bid-prices Un modello con intervalli disgiunti, ciascuno dedicato a una sola classe, è **statico**; se le richieste sono interleaved nel tempo è **dinamico**. La policy si esprime tramite **protection levels** oppure **bid-prices** (prezzo minimo accettabile per vendere un posto).

10.13 Modello statico con segmentazione perfetta della domanda

Assunzioni, stadi e stato Le domande sono indipendenti e avvengono sequenzialmente (prima D_n); nessun cliente cambia classe. Gli stadi decisionali sono $j = n, n-1, \dots, 1$. La variabile di stato è la **capacità residua** s_j , con stato iniziale $s_n = C$. Condizione al contorno:

$$V_0(s_0) = 0, \quad s_0 = 0, 1, \dots, C.$$

Ricorsione DP in forma scambiata La ricorsione assume la forma scambiata:

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

Con uno shift di indice e notazione compatta:

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

10.14 Valore marginale atteso e livelli di protezione

Valore marginale atteso della capacità Si definisce:

$$\Delta V_j(s) := V_j(s) - V_j(s-1),$$

che misura il **costo opportunità** di un posto dato s .

Somma telescopica e riscrittura Si ottiene:

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

La riscrittura usa una **somma telescopica**:

$$V_j(s-x) = V_j(s) - \sum_{z=1}^x \Delta V_j(s+1-z).$$

Proprietà dei valori marginali Si possono provare:

$$\Delta V_j(s+1) \leq \Delta V_j(s), \quad \Delta V_{j+1}(s) \geq \Delta V_j(s), \quad \forall s, j.$$

Regola di accettazione e protezione annidata I termini $p_{j+1} - \Delta V_j(s+1-z)$ sono decrescenti in z : si aumenta x finché compare il primo termine negativo o si raggiunge $\min\{s, D_{j+1}\}$. Si ottiene un livello di protezione **annidato**:

$$y_j^* := \max\{y : p_{j+1} < \Delta V_j(y)\}, \quad j = 1, \dots, n-1. \quad (18)$$

La decisione ottima allo stadio $j+1$ è:

$$x_{j+1}^*(s_{j+1}, D_{j+1}) = \min\{(s_{j+1} - y_j^*)^+, D_{j+1}\}.$$

Lo scambio tra aspettazione e ottimizzazione è giustificato dal fatto che non serve conoscere D_{j+1} in anticipo: si accetta finché si raggiunge la protezione o si esaurisce la capacità o termina lo stadio.

10.15 Modello dinamico con segmentazione perfetta della domanda

Assunzioni e probabilità di arrivo Si rilassa la sequenzialità della domanda, mantenendo una segmentazione rigida. Il tempo è discretizzato in $t = 1, \dots, T$ assumendo **al più un arrivo per intervallo**. Sia $\lambda_j(t)$ la probabilità di arrivo della classe j nell'intervallo t , con vincolo:

$$\sum_{j=1}^n \lambda_j(t) \leq 1, \quad \forall t.$$

Funzione valore e condizioni al contorno Si determinano $V_t(s)$ (capacità residua s) con:

$$V_t(0) = 0, \quad t = 1, \dots, T, \quad V_{T+1}(s) = 0, \quad s = 0, 1, \dots, C.$$

Ricavo disponibile e decisione binaria Sia $R(t)$ il **ricavo disponibile** al tempo t : vale p_j se arriva un cliente di classe j , 0 altrimenti. Alla richiesta si decide con $x \in \{0, 1\}$ se accettare o meno.

Ricorsione DP in forma scambiata La ricorsione assume la forma scambiata:

$$V_t(s) = \mathbb{E} \left[\max_{x \in \{0,1\}} \{R(t)x + V_{t+1}(s-x)\} \right].$$

Usando i **valori marginali attesi** si ottiene una politica in termini di **protection levels**, qui potenzialmente **time-varying**.

10.16 Modello dinamico con scelta del cliente

Modello di scelta Se la segmentazione perfetta viene rilassata, occorre un **choice model**. La decisione di controllo al tempo t è il sottoinsieme $S_t \subseteq N = \{1, \dots, n\}$ delle classi offerte.

Ricavi e probabilità di scelta Si introduca $p_0 = 0$ per il caso di **non-acquisto**. Con probabilità λ arriva un potenziale passeggero; dato S_t , la probabilità di scelta della classe j è $P_j(S_t)$, includendo $P_0(S_t)$, con vincoli:

$$P_j(S) \geq 0, \quad S \subseteq N, \quad j \in S \cup \{0\}, \quad \sum_{j \in S} P_j(S) + P_0(S) = 1, \quad S \subseteq N.$$

Probabilità di acquisto dipendenti dalla decisione Le probabilità dipendenti dalla decisione sono $\lambda P_j(S_t)$ per $j = 1, \dots, n$, e $(1 - \lambda) + \lambda P_0(S_t)$ per $j = 0$.

Ricorsione DP In questo caso la decisione deve essere dichiarata prima della scelta del passeggero, quindi la ricorsione è nella forma standard:

$$V_t(s) = \max_{S_t \subseteq N} \left\{ \sum_{j \in S_t} \lambda P_j(S_t) (p_j + V_{t+1}(s - 1)) + (\lambda P_0(S_t) + 1 - \lambda) V_{t+1}(s) \right\}.$$

Osservazione conclusiva La forma della ricorsione riflette la **struttura informativa**: qui la massimizzazione è sull'**aspettazione** perché la decisione precede la scelta del cliente.

11 CAP 10 – Programmazione dinamica numerica per stati discreti

11.1 Catene di Markov a tempo discreto

Definizione di catena di Markov a tempo discreto Una **catena di Markov a tempo discreto** è un processo stocastico con variabile di stato s_t , $t = 0, 1, 2, \dots$, che assume valori in un insieme **discreto**. Se lo spazio degli stati è numerabile, è possibile associare ciascuno stato a un numero intero e rappresentare il processo mediante una **rete**.

Probabilità di transizione Le transizioni tra stati sono rappresentate da archi diretti etichettati con le **probabilità di transizione**. Le probabilità di transizione sono **probabilità condizionate** che dipendono solo dallo stato corrente:

$$\pi(i, j) := \mathbb{P}\{s_{t+1} = j \mid s_t = i\}.$$

Catene omogenee Nel caso di problemi a **orizzonte infinito** si utilizzano catene **omogenee** (o **time-invariant**), in cui le probabilità di transizione non dipendono dal tempo.

Matrice di transizione Una catena di Markov a tempo discreto può essere descritta raccogliendo le probabilità di transizione in una **matrice di transizione a un passo** Π , in cui l'elemento π_{ij} rappresenta la probabilità di transizione dallo stato i allo stato j .

Proprietà di normalizzazione Poiché dopo una transizione il processo deve necessariamente trovarsi in uno stato dello spazio degli stati, ogni riga della matrice di transizione somma a uno:

$$\sum_{j=1}^N \pi(i, j) = 1, \quad \forall i.$$

11.2 Catene di Markov controllate e MDP

Azioni e transizioni controllate Nei **processi decisionali markoviani** (MDP), le transizioni sono parzialmente controllate tramite la selezione di azioni. In ciascuno stato i è disponibile un insieme finito di azioni ammissibili $A(i)$. Per ogni azione $a \in A(i)$ sono definite probabilità di transizione:

$$\pi(i, a, j).$$

Distribuzione stazionaria Per valutare le prestazioni di una politica di controllo stazionaria, si può considerare la **probabilità di lungo periodo** di trovarsi in ciascuno stato, indicata con $q(i)$ e raccolta nel vettore q . Tali probabilità potrebbero non esistere se la catena non soddisfa opportune proprietà strutturali.

Proprietà strutturali delle catene Una catena può presentare stati **transienti**, **ricorrenti**, **assorbenti** e **periodicità**. Si assume una **unichain**, tale che ogni stato possa essere visitato infinitamente spesso nel lungo periodo e raggiunto da ogni altro stato in tempo finito con probabilità positiva. Nel seguito si assume che il processo di Markov sia **ben comportato** per ogni politica di controllo.

11.3 MDP a orizzonte temporale finito

Ricorsione di programmazione dinamica Per un MDP a **orizzonte finito**, la ricorsione di DP è:

$$V_t(i) = \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_j \pi(i, a, j) V_{t+1}(j) \right\}, \quad i \in S.$$

Cosa insegna l'esempio (arresto ottimo su random walk) L'esempio mostra che, in un problema di arresto ottimo a orizzonte finito, la politica risultante può essere **non stazionaria** e dipendere dal tempo residuo (confronto tra **wait** e **stop** tramite DP).

11.4 Processi decisionali markoviani a orizzonte infinito

Equazioni di Bellman a orizzonte infinito Nel caso di un MDP a **orizzonte infinito**, la funzione valore è definita implicitamente da equazioni del tipo:

$$V(i) = \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_j \pi(i, a, j) V(j) \right\}, \quad i \in S, \quad (2)$$

oppure, quando il contributo immediato dipende anche dallo stato successivo,

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

Rappresentazione vettoriale Per MDP finiti, lo spazio degli stati è:

$$S = \{1, \dots, n\},$$

e la funzione valore $V : S \rightarrow \mathbb{R}$ è rappresentabile come vettore $V \in \mathbb{R}^n$ con componenti $V(i)$.

Strategie di soluzione Esistono due strategie fondamentali:

- **Value iteration**: iterazioni economiche, convergenza al valore ottimo solo nel limite;
- **Policy iteration**: iterazioni più costose, ma convergenza in tempo finito per MDP finiti (numero finito di politiche).

11.5 Operatori di Bellman

Operatore ottimo Data una funzione valore generica \tilde{V} , si definisce l'operatore T :

$$[T\tilde{V}](i) = \text{opt}_{a \in A(i)} \sum_{j \in S} \pi(i, a, j) \{ h(i, a, j) + \tilde{V}(j) \}, \quad i \in S. \quad (4)$$

Operatore associato a una politica Data una politica stazionaria generica μ , si definisce l'operatore T_μ :

$$[T_\mu \tilde{V}](i) = \sum_{j \in S} \pi(i, \mu(i), j) \left\{ h(i, \mu(i), j) + \tilde{V}(j) \right\}, \quad i \in S. \quad (5)$$

Ruolo degli operatori e punti fissi L'operatore T è centrale per la **value iteration**, mentre T_μ è centrale per la **policy iteration**. Il vettore valore ottimo V è un **punto fisso** di T :

$$V = TV. \quad (6)$$

La funzione valore V^μ di una politica stazionaria μ è il punto fisso di T_μ .

Condizioni di esistenza Per un MDP finito con **sconto stretto** ($\gamma < 1$) e contributi per stadio **limitati** (esiste M con $|h(i, a, j)| \leq M$), si assume che T e T_μ siano **operatori di contrazione**, con punto fisso unico.

Miglioramento delle politiche L'operatore T fornisce una caratterizzazione della politica stazionaria ottima e consente di **migliorare** una politica non ottima usando il valore V^μ , che viene ottenuto tramite T_μ .

11.6 Value iteration

Idea di base Con sconto stretto ($\gamma < 1$), si cerca un punto fisso di T tramite iterazione:

$$V^{(k+1)} = TV^{(k)}.$$

Se $V^{(k)} \rightarrow V$, allora V è un punto fisso di T e rappresenta la soluzione.

11.7 Algoritmo di Value Iteration

Algorithm 1 Value Iteration per MDP a orizzonte infinito

```
1: Scegliere una funzione valore iniziale  $V^{(0)}$  (ad esempio  $V^{(0)}(i) = 0$  per ogni  $i \in S$ )
2: Scegliere una tolleranza  $\varepsilon$ 
3: Inizializzare  $k = 0$  e stop=false
4: while stop  $\neq$  true do
5:   for ogni stato  $i \in S$  do
6:     
$$V^{(k+1)}(i) = \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_{j \in S} \pi(i, a, j) V^{(k)}(j) \right\}$$

7:   end for
8:   if  $\|V^{(k+1)} - V^{(k)}\|_1 < \varepsilon$  then
9:     stop=true
10:  else
11:     $k = k + 1$ 
12:  end if
13: end while
14: Porre  $\hat{V} = V^{(k+1)}$ 
15: Per ogni  $i \in S$ , determinare una politica stimata ottima:
```

$$\hat{\mu}(i) \in \arg \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_j \pi(i, a, j) \hat{V}(j) \right\}$$

```
16: Restituire  $\hat{V}$  e  $\hat{\mu}$ 
```

Cosa insegna l'esempio numerico (value iteration) L'esempio illustra che la politica ottima ottenuta da value iteration può dipendere in modo sensibile da probabilità di transizione e fattore di sconto, e che la convergenza in iterazioni può variare significativamente con tali parametri.

11.8 Policy iteration

Motivazione Con value iteration può accadere di individuare presto la politica ottima (da valori approssimati) ma senza valutarne ancora con precisione il valore. Policy iteration mira invece alla valutazione (più) diretta del valore di una politica.

Policy evaluation come sistema lineare Per una politica stazionaria μ , T_μ non include ottimizzazione:

$$[T_\mu \tilde{V}](i) = f(i, \mu(i)) + \sum_{j \in S} \pi(i, \mu(i), j) \tilde{V}(j), \quad i \in S.$$

Il valore V^μ soddisfa:

$$V^\mu(i) = f(i, \mu(i)) + \sum_{j \in S} \pi(i, \mu(i), j) V^\mu(j), \quad i \in S.$$

Definendo la matrice di transizione indotta Π^μ e il vettore dei contributi f^μ :

$$f^\mu := \begin{bmatrix} f(1, \mu(1)) \\ f(2, \mu(2)) \\ \vdots \\ f(n, \mu(n)) \end{bmatrix}, \quad (9)$$

si ottiene il sistema:

$$V^\mu = f^\mu + \gamma \Pi^\mu V^\mu, \quad (I - \gamma \Pi^\mu) V^\mu = f^\mu.$$

Formalmente:

$$V^\mu = (I - \gamma \Pi^\mu)^{-1} f^\mu.$$

Per matrici grandi e sparse, metodi diretti possono essere proibitivi; si adottano quindi metodi iterativi.

Policy improvement Data V^μ , una politica migliorata $\tilde{\mu}$ è definita da:

$$\tilde{\mu}(i) \in \arg \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_j \pi(i, a, j) V^\mu(j) \right\}, \quad i \in S.$$

Per un problema di massimizzazione:

$$V^\mu(i) \leq V^{\tilde{\mu}}(i), \quad i \in S,$$

(con disuguaglianza invertita in minimizzazione). Se μ non è ottima, la disuguaglianza è stretta per almeno uno stato. Poiché il numero di politiche stazionarie deterministiche è finito, una sequenza di miglioramenti porta a una politica ottima.

11.9 Algoritmo di Policy Iteration

Algorithm 2 Policy Iteration per MDP finito a orizzonte infinito

1: Definire una politica stazionaria iniziale arbitraria $\mu^{(0)}$

2: Inizializzare $k = 0$ e **stop**=false

3: **while** **stop** \neq true **do**

4: **Policy evaluation:** risolvere

$$(I - \gamma \Pi^{\mu^{(k)}}) V^{\mu^{(k)}} = f^{\mu^{(k)}}$$

5: **Policy improvement:** porre, per ogni $i \in S$,

$$\mu^{(k+1)}(i) \in \arg \text{opt}_{a \in A(i)} \left\{ f(i, a) + \sum_j \pi(i, a, j) V^{\mu^{(k)}}(j) \right\}$$

6: **if** $\mu^{(k+1)} = \mu^{(k)}$ **then**

7: **stop**=true

8: **else**

9: $k = k + 1$

10: **end if**

11: **end while**

12: Restituire la funzione valore ottima e la politica stazionaria ottima

Cosa insegna l'esempio numerico (policy iteration) L'esempio evidenzia che policy iteration può raggiungere la politica ottima in un numero finito (spesso piccolo) di passi di miglioramento, grazie alla valutazione esplicita della politica corrente.

11.10 Value iteration vs. policy iteration

Confronto e collegamento Value iteration impiega molte iterazioni economiche e converge nel limite; policy iteration usa poche iterazioni potenzialmente costose e converge in tempo finito per MDP finiti. La valutazione di una politica può anche essere eseguita iterativamente tramite:

$$V_\mu^{(k+1)}(i) = f(i, \mu(i)) + \sum_{j \in S} \pi(i, \mu(i), j) V_\mu^{(k)}(j), \quad i \in S, \quad (10)$$

e interrompendo in modo prematuro tale procedura si ottiene una stima \widehat{V}_μ da usare nel miglioramento, dando luogo a **optimistic policy iteration** e, più in generale, a metodi di **generalized policy iteration**. Value iteration e policy iteration possono quindi essere visti come estremi di un continuum di approcci.

11.11 Collegamento con Reinforcement Learning

DP model-free e Q-factors Quando le probabilità di transizione (e possibilmente i contributi immediati) non sono note, si passa a una DP **model-free**. In tale contesto, la funzione valore è tipicamente sostituita dai **Q-factors** $Q(s, a)$ dipendenti da stati e azioni.

Q-learning e SARSA Il **Q-learning** è il corrispettivo RL della value iteration ed è un metodo **off-policy**. I corrispettivi RL della policy iteration sono **on-policy**; un approccio noto è **SARSA**.

Osservazione operativa in RL In RL non è in generale possibile valutare esattamente il valore di una politica se ciò richiede simulazioni Monte Carlo costose o esperimenti online; è quindi necessario effettuare un passo di miglioramento prima o poi, generando diverse varianti di strategie.

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

13 CAP 12 – Simulazione

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