CAP 1
INTRODUZIONE ALLE DECISIONI IN CONDIZIONE DI INCERTEZZA

Le **decisioni** sotto ***incertezza***, rendono insufficiente la previsione puntuale e richiedendo *modelli*.

Obiettivi

- distinguere **ottimizzazione robusta** e **stocastica**
- introdurre i principali **modelli decisionali**
- rappres *decisioni adattive* con **alberi decisionali**
- chiarire **multistadio** vs **multiperiodo**
- valutare **scenari**, **stabilità** *in-sample* e *out-of-sample*

COSTO INCERTEZZA–MODELLI DECISIONALI

Una **prev puntuale** *non è sufficien*: conta il *costo dell'errore* e il *criterio decisionale*.

Costo dell'errore di previsione. Il criterio di prev dipende dalla *f di perdita*.

- **errore quadratico medio (MSE)**: penalità simmetrica, previsione ottima pari al **valore atteso** $MSE(x) \doteq \mathbb{E}\left[(X-x)^2\right] = \mathbb{E}[X^2] - 2x\,\mathbb{E}[X] + x^2$
- **deviazione assoluta**: penalità lineare, previsione ottima pari alla **mediana** o a un **quantile** $\mathbb{E}\left[|X-x|\right]$

Ottimizzazione sotto incertezza. L'obb non è prevedere, ma scegliere decisioni ottimali in presenza di fattori aleatori.

- **worst-case robust**: ottimizzazione sullo *scenario peggiore*, senza distribuzioni di probabilità $\min_{x\in S} \max_{\xi\in U} f(x,\xi)$
- **stocastica**: *minimizzazione del valore atteso*, con modellazione probabilistica esplicita $\min_{x\in S} \mathbb{E}_{\mathbb{P}}[f(x,\tilde{\xi})]$

ALBERI DECISIONALI

Rappresentazione temporale. Gli *alberi decisionali* descrivono in modo esplicito la ***sequenza di decisioni e realizzazioni aleatorie***, con *decisioni adattive* nel *tempo*.

Struttura dei nodi.

- ***nodi decisionali***: scelte tra *alternative esclusive*
- **nodi casuali**: esiti aleatori con *probabilità associate*

Procedura di soluzione. La valutazione avviene per *backward induction*: *un nodo è valutabile dopo i successori*.

EVPI & VSS

Valori ottimi. Descrivono diversi livelli di *informazione* e di *modellazione dell'incertezza*.

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

- **f***: ottimo stocastico *here-and-now*
- **fpi***: ottimo con info perfetta *wait-and-see*
- **fev***: ottimo deterministico a *valori medi*
- **feev**: costo atteso della soluzione EV

Misure del valore dell'info. Quantificano il *beneficio di modellare o osservare l'incertezza*.

- **EVPI**: beneficio *teorico della chiaroveggenza*
- **VSS**: *guadagno della soluzione stocastica*

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

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

$$VSS = f_{EEV} - f^* \quad EVPI = f^* - f^*_{PI}.$$

MODELLI A DUE STADI

Idea. Le decisioni sono separate in here-and-now e wait-and-see, adattandosi tramite il ricorso. $\mathbf{Q}(x) = \mathbb{E}_{\mathbb{P}}\left[Q(x,\tilde{\xi})\right]$

Fattibilità. Il *1 stadio* è *ammissibile solo se il 2 è fattibile per ogni scenario* (complete relatively complete recourse).

$$\min_x c^T x + \mathbf{Q}(x) \quad \text{s.t. } Ax = b,\ x \geq 0,$$

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

ATO

Ambiente produttivo. I prodotti finali sono assemblati da componenti comuni dopo l'osservazione della domanda.

Struttura decisionale.

1. **here-and-now**: *produzione*
2. **wait-and-see**: *assemblaggio*

Modello. Problema stocastico a due stadi con *valore di ricorso*

$$\max \quad -\sum_{i\in[n_i]} C_i x_i + \sum_{s\in[n_s]} \pi^s \left(\sum_{j\in[n_j]} P_j y_j^s\right)$$

$$\text{s.t.} \quad \sum_{i\in[n_i]} T_{im} x_i \leq L_m, \qquad m \in [n_m]$$

$$y_j^s \leq d_j^s, \qquad j\in[n_j],\ s\in[n_s]$$

$$\sum_{j\in[n_j]} G_{ij} y_j^s \leq x_i, \qquad i\in[n_i],\ s\in[n_s]$$

$$x_i, y_j^s \in \mathbb{Z}_+, \qquad i\in[n_i] j\in[n_j],\ s\in[n_s].$$

PLANT LOCATION MODEL

$$\min \quad \sum_{i\in\mathcal{P}} f_i y_i + \sum_{s\in S} \pi^s \left(\sum_{i\in\mathcal{P}}\sum_{j\in\mathcal{D}} c_{ij} x_{ij}^s + \sum_{j\in\mathcal{D}} \beta_j z_j^s\right)$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{P}} x_{ij}^s + z_j^s = d_j^s \quad \forall j\in\mathcal{D},\ \forall s\in S$$

$$\sum_{j\in\mathcal{D}} x_{ij}^s \leq R_i y_i \quad \forall i\in\mathcal{P},\ \forall s\in S$$

$$x_{ij}^s \geq 0,\ z_j^s \geq 0,\ y_i \in \{0,1\}.$$

Idea. La scelta di apertura degli impianti è presa sotto incertezza, mentre i flussi di trasporto sono adattati dopo l'osservazione della domanda.

NEWSVENDOR

Idea. Le **quantità produttive** sono decise **progressivamente sotto incertezza** e gli esiti finali generano costi di overage e underage.

$$\min \quad \sum_{n\in\mathcal{N}_2} \pi^n \sum_{i\in\mathcal{I}} (c_i^o o_i^n + c_i^u u_i^n)$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{I}} x_i^0 \leq K_1$$

$$m_i \delta_i^0 \leq x_i^0 \leq K_1 \delta_i^0 \quad \forall i\in\mathcal{I}$$

$$\sum_{i\in\mathcal{I}} x_i^n \leq K_2 \quad \forall n\in\mathcal{N}_1$$

$$m_i \delta_i^n \leq x_i^n \leq K_2 \delta_i^n \quad \forall i\in\mathcal{I},\ \forall n\in\mathcal{N}_1$$

$$x_i^0 + x_i^{a(n)} = d_i^n + o_i^n - u_i^n \quad \forall i\in\mathcal{I},\ \forall n\in\mathcal{N}_2$$

$$x_i^n \in \mathbb{Z}_+,\ \delta_i^n \in \{0,1\} \quad \forall i\in\mathcal{I},\ \forall n\in\{0\}\cup\mathcal{N}_1$$

$$u_i^n,\ o_i^n \geq 0 \quad \forall i\in\mathcal{I},\ \forall n\in\mathcal{N}_2.$$

UNIT COMMTMENT

Modello multiperiodale.

- decisioni di **attivazione** *here-and-now*
- **produzione** adattata alla *domanda*

$$\min \quad \sum_{i\in[I],t\in[T]} (E_i u_{it} + F_i s_{it}) + \sum_{\omega\in\Omega} \pi^\omega \sum_{i\in[I],t\in[T]} C_i\,(q_{it}^\omega - m_i u_{it})$$

$$\text{s.t.} \quad \sum_{i\in[I]} q_{it}^\omega \geq d_t(\omega) \quad \forall t\in[T],\ \forall\omega\in\Omega$$

$$m_i u_{it} \leq q_{it}^\omega \leq M_i u_{it} \quad \forall i\in[I],\ \forall t\in[T],\ \forall\omega\in\Omega$$

$$s_{it} \geq u_{it} - u_{i,t-1} \quad \forall i\in[I],\ \forall t\in[T]$$

$$u_{it} \leq a_i \quad \forall i\in[I],\ \forall t\in[T]$$

$$u_{it} \in \mathbb{Z}_+,\ s_{it}\in\mathbb{Z}_+,\ q_{it}^\omega \geq 0 \quad \forall i\in[I],\ \forall t\in[T],\ \forall\omega\in\Omega.$$

CAP 2
ELEMENTI DI COMPLESSITÀ COMPUTAZIONALE

Classificazione dei *prob di ottimizzazione* in base alla **difficoltà intrinseca del prob**, *indipendentemente* dall'**algo**.

Obiettivi principali
- distinguere **complessità** del *prob* vs *algo*
- introdurre **prob di decisione vs ottimizzazione**
- definire le classi **P**, **NP**, **NPH**, **NPC**

COMPLESSITÀ INTRINSECA: SCHEDULING

Problema di scheduling (min,max) su macchina singola. Sequenziamento di job su una macchina per controllare il max ritardo rispetto alle due date.

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

$L_{\max} \doteq \max_{j\in[n]} L_j \quad L_j \doteq C_j - d_j.$

Teorema (regola EDD – Earliest Due Date). Per il prob *1\\Lmax* esiste una *sol ottima* in cui i job sono ordinati per due date crescenti. $d_{\sigma(k)} \leq d_{\sigma(k+1)}$

➡ *algo polinomiale*

➡ *prob computazionalmente trattabile*

Realise Time. L'introduzione dei tempi di rilascio (*1\ri\Lmax*) vincola l'avvio dei job e rende non più ottima la regola EDD, aumentando la compl intrinseca del prob.

Non esistono algo di compl polinomiale che lo risolvono, solo **branch-and-bound**.

PROBLEMI DI DECISIONE VS OTTIMIZZAZIONE

- **prob di decisione (PD)**: risposta binaria *(sì / no)*
- **prob di ottimizzazione (PO)**: ricerca migliore sol rispetto a una f obiettivo $\min_{x\in S} f(x)$

Legame tra PO e PD. Dato un PO, si definisce un PD scegliendo un valore k e chiedendo se esiste x in S tc f(x)<k.

PD→PO. Se si ha a disposizione un algo efficiente per PO, è possibile risolvere in modo efficiente PD

➡ *se PD è difficile ⇒ PO non può essere facile*

➡ *per dimostrare che un prob di ottimizzazione è difficile, è sufficiente dimostrare che è difficile il corrispondente prob di decisione*

SCHEDULING CON RELEASE TIMES

La versione decisionale del prob di scheduling 1/rj/Lmax è **NP-completa**.
- *mostra che l'intrattabilità nasce con l'introduzione dei tempi di rilascio*
- *PO è NP-difficile*

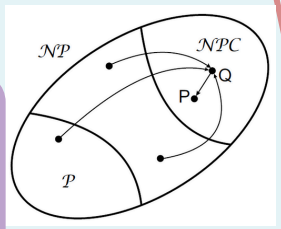
TEORIA DELLA NP-COMPLETEZZA

Esiste una vasta classe di *prob di ottimizzazione* per cui non sono noti algo polinomiali.

Idea centrale. La **teoria della NP-completezza** mostra che molti di questi prob sono equivalenti dal punto di vista *computazionale*.

➡ *se uno solo ammettesse un algo pol tutti i prob della classe lo ammetterebbero*

Conseguenza fondamentale. Decenni di ricerca senza successo suggeriscono che
- *tali algo probabilmente non esistono*
- *la difficoltà è intrinseca, non dovuta a modelli*



CLASSI NPH – NPC

Classe NPH. Prob P tc ogni prob in NP è riduc a P.

➡ *PO + PD*

Classe NPC. Prob P tc
- P è in NP **e** P è NPH

➡ *problemi più difficili in NP, tutti equivalenti*

➡ *per dimostrare che un PD P è NPC, occorre dimostrare che P è in NP e un prob NP-completo Q può essere ridotto in tempo polinomiale a P*

Teorema di Cook. Il prob della soddisfacibilità booleana è NP-completo.

$$(A \text{ or } B) \text{ and } (\text{not}(A) \text{ or } C)$$

CLASSI P – NP

Classe P. *Prob di decisione* per cui esiste un algo di compl polinomiale: *il numero di passi è limitato superiormente da una funzione polinomiale*. L'**algo**
- *trova una soluzione*
- *ne verifica la correttezza*

Classe NP. *Prob di decisione* le cui istanze che hanno risposta positiva sono verificabili in tempo polinomiale. Esiste un certificato polinomiale su un ***calcolatore non deterministico***.

RIDUZIONE POLINOMIALE

Un prob P è **riducibile** a Q se *ogni istanza di P può essere trasformata in tempo polinomiale in un'istanza di Q* con la stessa risposta.
- se P è difficile e **P<Q** ⇒ Q non può essere facile
- *la compl di P non è maggiore della compl di trasformare P in Q e poi risolvere Q*

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

IMPATTO CODIFICA (KNAPSACK)

Idea chiave. La compl dipende dalla codifica dell'input, non solo dal prob.
- *B* è **codificato in binario** ⇒ input di dimensione logB
- *O(nB)* è **esponenziale** nella dimensione dell'input

Pseudo-Polinomiale. L'algo, rispetto alla codifica binaria, ha *compl esponenziale*. Se si utilizzasse un **computer con una codifica unaria**, l'algo avrebbe *compl polinomiale*.

CAP 3
METODI DI DECOMPOSIZIONE IN OTTIMIZZAZIONE

MANCA

MANCA

MANCA

MANCA

- **MANCA**

$\min_{x \in S} f(x)$

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA

$(A \text{ or } B) \text{ and } (\text{not}(A) \text{ or } C)$

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA

CAP 4

SISTEMI MRP – ERP – APPROCCIO JIT

Classificazione dei *sis di pianif* e *controllo della produz multilivello e con variabilità.*

Obiettivi principali

- distinguere **logiche push e pull**
- chiarire il ruolo di **variabilità**, **WIP** e **lead time**
- pianificazione **MRP** a *capacità infinita*
- approccio **Just-In-Time** (*Toyota*)

Idea chiave. Le *prestazioni* del sis produttivo dipendono dalla **variabilità** (propagata o controllata).

LIMITI ED EVOLUZIONE DEI SISTEMI MRP

Limiti degli approcci classici. I modelli tradizionali risultano inadeguati

- in *ambienti non make-to-stock (make-to-order, assemble-to-order)*
- presenza di *vincoli di capacità produttiva*
- *distinte base multilivello*

Effetto di amplificazione della variabilità. La **propagazione dei fabbisogni** lungo la distinta base può generare una *forte amplificazione della variabilità*, anche con domanda finale regolare.

Evoluzione dei modelli MRP. I sis MRP (***Material Requirements Planning***) nascono come ***risposta operativa al prob del lot-sizing multilivello***

➡ assunzione di *capacità infinita*

➡ utilizzo di *lead time fissati a priori*

Evoluzione verso MRPII ed ERP.

- **MRPII** (***Manufacturing***): introduce la *verifica dei vincoli di capacità*
- **RCCP** (***Rough Cut Capacity***): *verifica* aggregata e approssimata della *capacità*
- **CRP** (***Capacity Requirement***): *verifica dettagliata capacità* sulle singole risorse
- **ERP** (***Enterprise Resource***): integrazione *pianific con f commerciali e finanziarie*

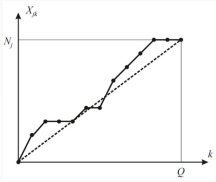
APPROCCIO JUST-IN-TIME

Definizione e obiettivo. Il *Just-In-Time (JIT)* mira a *ridurre la variabilità* alla fonte tramite *produzione livellata* (**production smoothing**), *lotti piccoli* e *frequenti* e ***riduzione dei tempi di setup***, con l'obiettivo di contenere WIP e lead time.

Logiche di controllo.

- **push**: *rilascio ordini da previsione*
- **pull**: produzione attivata *da domanda* reale
- **kanban**: *controllo pull locale a segnali*
- **CONWIP**: controllo *pull* con *WIP* globale

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



Goal chasing. Il *Toyota Goal Chasing* seleziona la ***sequenza produttiva che rende regolare il consumo dei componenti***, minimizzando la distanza tra consumo ideale e consumo effettivo lungo il ciclo.

Rotazione ciclica dei prodotti. I prodotti si *alternano* su una linea con periodo di rotazione. $p_i T_i = d_i T_c \quad \Rightarrow \quad T_i = \frac{d_i}{p_i} T_c$

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

Il ***limite inferiore*** dipende dai *tempi di setup* e dal *rapporto tra tassi di domanda e produzione*, evidenziando un legame con i fenomeni di congestione.

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

LOGICA MRP

Assunzione di capacità infinita. Il *vincolo di capacità* non è modellato ed è surrogato da *lead time fissati a priori*.

Lead time offsetting. Gli *ordini pianificati* sono anticipati nel tempo rispetto ai fabbisogni.

Record MRP. Per ogni codice e periodo

- *fabbisogni lordi*
- *magazzino* disponibile (*on-hand*)
- *ordini* emessi (*on-order*)
- *fabbisogni netti*
- *ordini pianificati*

➡ la domanda dei prodotti finiti è definita dal **MPS** (***Muster Production Schedule***)

➡ l'**MRP** procede ricorsivamente lungo la distinta base

Ordini pianificati. Non sono esecutivi; al rilascio diventano ordini operativi e allocano giacenze.

Lot-sizing. Regola base *lot-for-lot*: produci esattamente ciò di cui hai bisogno.

LEGGE DI LITTLE

Prestazioni di shop floor. La **Factory Physics** descrive le prestazioni tramite *throughput*, *flow time* e *WIP*.

Legge di Little. Esprime il *legame strutturale* tra queste grandezze. $\text{WIP} = \text{throughput} \times \text{flow time} \quad L = \lambda\,(W_q + t_s)$

Modello a singola macchina e variabilità. In una singola macchina, l'*attesa in coda cresce con l'utilizzazione* e con la *variabilità* dei tempi di interarrivo e servizio. $u = \lambda/\mu$

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

Buffering law. In presenza di variabilità, il sistema deve introdurre *buffer sotto forma di WIP*, *capacità* o tempo o *lead time*.

NERVOSISMO

Nervosismo. *Piccole variazioni nel MPS producono grandi variazioni negli ordini pianificati dovute a*

- **lot-sizing** a *quantità variabile*
- **effetto di bordo**: instabilità da *rolling horizon*

Effetti. *Instabilità* del piano e *ordini urgenti.*

Mitigazione.

- **time fencing**: *congelamento* temporale MPS
- **firm planned orders**: ordini *non modificabili*

MPS e CRP

MPS. Il MPS è l'*input primario dell'MRP*, basato su *ordini cliente* e **forecasting**; può essere validato tramite **RCCP** e strutturato a due livelli in contesti ATO.

CRP. Il CRP *verifica a posteriori la capacità*; la correzione manuale è complessa e può generare lead time gonfiati e WIP, innescando un *circolo vizioso*.

CAP 5
SCHEDULAZIONE in PRODUZ-SERVIZI

Schedulazione di job su risorse nel tempo, con *vincoli tecnologici e di capacità*.

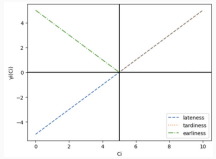
Obiettivi principali

- **misure di prestazione**: f sui tempi di completamento, aggregate min-sum o min-max
- **classificazione dei prob**: *notazione di Graham*
- **compl computaz**: distinzione tra casi polinomiali (EDD, WSPT, Johnson) e prob NPH
- **strategie di soluzione**: uso *euristiche* e *shifting bottleneck* per decomporre *sis complessi*

MISURE DI PRESTAZIONE

Funzioni di penalità.

- **tempo di completamento** (*Ci*): istante di *fine dell'ultima operazione del job*
- **flow time** (*Fi*): Ci – ri, *tempo totale* trascorso nel sistema
- **lateness** (*Li*): Ci – di, *anticipo o ritardo rispetto alla due date*
- **tardiness** (*Ti*): max(Ci – di, 0), penalizza solo i *ritardi*
- **earliness** (*Ei*): max(di – Ci, 0), penalizza solo gli *anticipi*
- **indicatore di ritardo** (*Ui*): vale 1 se Ci > di, 0 altrimenti



Misure aggregate. Flow time totale, flow time totale pesato, massima lateness, tardiness totale pesata, makespan (*massimo dei Ci*), numero di job in ritardo.

Soluzioni equivalenti. Una sol ottima rispetto a una misura è ottima anche per un'altra; **es** lateness totale e flow time totale differiscono solo per una costante.

Misure regolari. F non decrescenti dei tempi di completamento Ci.

Misure non regolari. F non monotone in Ci, con penalità di earliness e tardiness.

- **schedul semiattiva**: ogni op è eseguita il più presto possibile
- **schedul attiva**: non esiste op anticipabile senza ritardarne un'altra

Notazione di Graham (alpha | beta | gamma). Layout delle *macchine*, *vincoli* aggiuntivi, misura di *prestazione*.

ALGORITMI DI SOL NELLO SCHEDULING

Algoritmi polinomiali (casi speciali).

- **EDD**: ordinamento per *due date crescenti*; risolve 1\\Lmax
- **WSPT**: ordinamento per *wi/pi* decrescente; risolve 1\\wiCi
- **Johnson**: per F2\\Cmax; la sol ottima usa la *stessa sequenza sulle 2 macchine*

Regola ATC (Apparent Tardiness Cost). Assegna priorità combinando **peso del job**, durata dell'operazione e urgenza rispetto alla due date.

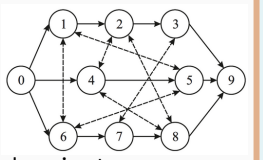
- se il ***job è in tempo*** la *priorità cresce* esponenzialmente
- se è in **ritardo** si riduce alla regola WSPT

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

Lookahead + ricerca locale.

- *beam search*: riduce la **miopia** delle regole di priorità
- *criticità*: evitare *minimi locali* (tabu, genetici), esplorare *grandi vicinati* (LNS), *evitare cicli*

Grafi disgiuntivi.



- **nodi**: operazioni + dummy iniziale/finale
- **archi congiuntivi**: precedenze tecnologiche del job
- **archi disgiuntivi**: capacità macchina (clique per macchina), da orientare
- **cammino critico**: lunghezza massima start→end = *makespan*

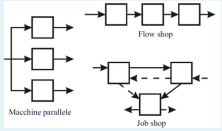
MACHINE SCHEDULING

Problemi di scheduling. Assegnazione di risorse a *job* nel *tempo*, rispettando ***vincoli*** *tecnologici*, di *capacità*, *precedenze*, ***tempi*** *di processo*, *due date*.

Soluzioni e diagrammi di Gantt. Una *sol* è definita dalle *sequenze di lavorazione sulle macchine* ed è visualizzata tramite *diagrammi di Gantt*, che *rappresentano graficamente l'allocazione temporale dei job*.

Tipi di flusso.

- **macchina singola**: una *sola* risorsa
- **macchine parallele**: identiche, correlate o scorrelate
- **flow shop**: *stesso ordine* di macchine
- **job shop**: cicli di lavorazione *diversi*
- **open shop**: *nessun ordine* prefissato



PROCEDURA SHIFTING BOTTLENECH

Idea. Affrontare il problema *J//Cmax* **decomponendolo** in una sequenza di *sottoproblemi su singola macchina*, sfruttando il grafo disgiuntivo.

Approssimazione del makespan. Ottenuta tramite *teste e code delle operazioni lungo il cammino critico*, che stimano i tempi di rilascio e le scadenze locali.

Riduzione. Ogni macchina induce un problema 1/ri/Lmax, risolto in modo efficiente.

Identificazione del collo di bottiglia. La macchina con Lmax peggiore; la sua sequenza viene fissata e il processo iterato sulle restanti macchine.

MODELLO MILP J//Cmax

Nel modello MILP per J//Cmax solo *perturbazioni degli archi disgiuntivi sul cammino critico sono utili, poiché evitano la creazione di cicli.*

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

CAP 6
PRICING

MANCA

MANCA

MANCA

MCNA

- **MANCA**

MANCA

MANCA

MANCA

MANCA

MANCA –

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA

CAP 7
PRINCIPIO DELLA DP

La **programmazione dinamica** è un principio per risolvere **problemi decisionali dinamici multistadio** tramite *decomposizione in sottoproblemi a singolo stadio*, con **decisioni adattive** in feedback (sis **Markoviano**).

Obiettivi

- **modellare** decisioni seq (open vs *closed-loop*)
- **Bellman**: *bilanciare costo attuale e valore futuro*
- gestire **stati**, **decisioni** e **incertezza**
- derivare **politiche ottime ricorsive**

PROBLEMI A ORIZZONTE INFINITO

Problema DP scontato. Il problema è formulato come ottimizzazione del valore atteso della somma infinita dei contributi, pesati da un *fattore di sconto* oppure, in alternativa, tramite il **contributo medio per stadio**.

$$\text{opt}_{\mu\in\mathcal{M}}\ \mathbb{E}_0\left[\sum_{t=0}^{T-1}\gamma^t f_t(s_t,\mu_t(s_t))+\gamma^T F_T(s_T)\right] \qquad \text{opt}\ \lim_{T\to\infty}\mathbb{E}_0\left[\frac{1}{T}\sum_{t=0}^{T-1} f(s_t,x_t)\right]$$

Politica decisionale. Una politica chiusa e non anticipativa associa a *ogni stato* una *decisione ammissibile* → sequenza di funzioni nel tempo. $x_t=\mu_t(s_t)\in X(s_t)$

- **politiche stazionarie** nei problemi a orizzonte infinito
- **politiche randomizzate** per *vincoli probabilistici* $\mathbb{P}\{s_t\in\bar{G}\}\geq 1-\alpha$

Modelli alternativi. Con domanda incerta si distinguono

- **vendite perse**: *domanda non soddisfatta eliminata*, penalità per unità persa
- **backlog**: *domanda accumulata*, con inventario **O** e arretrati **B** penalizzati

$$O_{t+1}=\max\{0,O_t-B_t+x_t-d_{t+1}\},\ B_{t+1}=\max\{0,-O_t+B_t-x_t+d_{t+1}\}$$

PROBLEMA DEL CAMMINO MINIMO

Rete diretta aciclica. Il prob è formulato su una rete diretta senza cicli, in cui ogni ***nodo*** rappresenta uno **stato** del sistema e ogni ***arco*** una possibile *transizione* con costo associato.

Approccio greedy. Una scelta locale basata sul **costo minimo immediato** non è in generale ottimale $\min_{j\in S_i}(c_{ij}+V_j)$

➡️per migliorare la decisione: misura della qualità dello stato successivo

Cammino minimo. Il cammino ottimo gode di una ***proprietà di annidamento***: ogni sottocammino di un cammino minimo è a sua volta ottimo.

Ricorsione di Bellman. Il valore di ciascun nodo è definito come ***minimo del costo dell'arco più il valore del nodo successore***.

- risoluzione tramite ordinamento topologico
- definizione della **value function** su tutti gli stati

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

PRINCIPIO DP

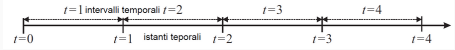
Dinamica del sistema. L'evoluzione del sis è descritta da una **legge di transizione di stato** che lega *stato corrente*, *decisione* e *fattori esogeni*. $s_{t+1}=g_{t+1}(s_t,x_t,\xi_{t+1})$

Variabili di stato.

- **fisiche**: influenzate dalle *decisioni*
- **informative**: *non* influenzate dalle decisioni
- **di credenza**: *non oggettive*

Politica non anticipativa. Decisioni dipendono dall'*info disponibile fino all'istante corrente*.

Scopo. Ottimizzare una f obiettivo additiva nel tempo.



EQUAZIONE DI BELLMAN

Regola miopica. Una decisione rapida consiste nel risolvere, nello stato corrente, un **problema a singolo stadio che ottimizza solo il contributo immediato**, trascurando gli effetti futuri.

Idea centrale. La DP introduce una ***value function*** che consente di bilanciare obiettivi di breve e lungo periodo e di ottenere la prestazione ottima.

Equazione di Bellman. Il *valore dello stato* è definito come *ottimo del contributo immediato più il valore atteso dello stato successivo*, parametrizzando il problema sullo stato.

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

Teorema di ottimalità. *Ogni sottoproblema generato lungo una traiettoria ottima ammette come soluzione la restrizione della politica ottima originale.*

ORIZZONTE FINITO SCONTATI

Funzione obiettivo. È il ***valore atteso della somma dei contributi nel tempo***. $\text{opt}\ \mathbb{E}_0\left[\sum_{t=0}^{t-1}\gamma^t f_t(s_t,x_t)+\gamma^T F_T(s_T)\right]$

- **contributo immediato**: costo o ricavo associato alla *decisione nello stato corrente*, eventualmente stocastico $f_t(s_t,x_t)=\mathbb{E}_t\left[h_t(s_t,x_t,\xi_{t+1})\right]$
- **contributo terminale**: valore assegnato allo *stato finale*

Fattore di sconto. Il parametro **γ∈(0,1)** pesa i contributi futuri e riflette la *preferenza temporale*, pur non essendo essenziale in orizzonte finito.

DP STOCASTICA PER ORIZZONTI FINITI

Risoluzione all'indietro. L'equazione di Bellman definisce un *problema statico ma non miopico,* risolto **procedendo backward dalla condizione terminale** e **costruendo ricorsivamente la value function** per ogni istante.

Uso delle funzioni valore. La conoscenza di *Vt()* guida le decisioni ottime.

- **caso deterministico**: sequenza di decisioni ottime e aggiornamento dello stato
- **caso stocastico**: simulazione *Monte Carlo* delle traiettorie

Orizzonte infinito. La DP scontata conduce a un'equazione funzionale in cui la value function è definita come **punto fisso** di un operatore. $V(s)=\text{opt}_{x\in X(s)}\left\{f(s,x)+\gamma\,\mathbb{E}\left[V\left(g(s,x,\xi)\right)\right]\right\}$ $V_{T-1}(s_{T-1})=\text{opt}_{x_{T-1}\in X(s_{T-1})}\left\{f_{T-1}(s_{T-1},x_{T-1})+\gamma\,\mathbb{E}\left[V_T\left(g_T(s_{T-1},x_{T-1},\xi_T)\right)\ \middle|\ s_{T-1},x_{T-1}\right]\right\}$

ALLOCAZIONE DISCRETA DI RISORSE

Knapsack. Selezione di un sottoinsieme di oggetti che *massimizza* il valore totale rispettando un vincolo di budget discreto, con decisioni binarie **tutto-o-niente**.

Riformulazione DP. *Non* è *dinamico* ⇒ allocazione sequenziale introducendo uno *stadio fittizio* ***k*** e usando il *budget residuo* come stato.

- **value function** *tabulata* su stati e stadi
- *ricorsione di Bellman* semplice e diretta
- algo a **compl pseudo-polinomiale** O(nB)

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

$$s_{k+1} = s_k - w_k x_k, \qquad s_1 = B.$$

$$\max \sum_{k=1}^{n} v_k x_k$$

$$\text{s.t.} \sum_{k=1}^{n} w_k x_k \le B,$$

$$x_k \in \{0,1\} \quad \forall k.$$

ALLOCAZIONE CONTINUA DI BUDGET

Formulazione iniziale. Il problema considera l'allocazione di un budget continuo tra attività, con *contributi al profitto descritti da funzioni fk()* crescenti e concave.

➡ f obb è concava

➡ ***hp* sol interne: Lagrangiana** dà condizioni di ottimalità necessarie e sufficienti

Riformulazione DP. Il problema non è dinamico ⇒ viene riformulato come *allocazione sequenziale introducendo uno stadio fittizio* ***k*** e usando il budget residuo come stato. La ***Vk(s) è infinito-dimensionale*** e richiede approssimazione.

- Vk(s) definita su **stato continuo**
- necessità di **discretizzazione e interpolazione**
- uso di **spline cubiche** per stimare valori fuori griglia
- ***risoluzione numerica tramite DP backward***

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

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

$$V_k(s_k) = \max_{0 \le x_k \le s_k} \{ f_k(x_k) + V_{k+1}(s_k - x_k) \}, \quad V_n(s_n) = \max_{0 \le x_n \le s_n} f_n(x_n) = f_n(s_n).$$

$$\max \sum_{k=1}^{n} f_k(x_k)$$

$$\text{s.t.} \sum_{k=1}^{n} x_k \le B,$$

$$x_k \ge 0 \quad \forall k.$$

LOT-SIZING DETERMINISTICO

Memorizzazione tramite tabella della Vk(s). L'approccio DP diretto è corretto ma diventa inefficiente *se lo spazio degli stati è grande o continuo* ⇒ **impraticabile la tabulazione**.

Struttura e riformulazione. Nel lot-sizing deterministico con soli *costi fissi di ordine* e *costi lineari di giacenza*, esiste una **forte proprietà strutturale** che consente di ridurre drasticamente il problema.

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

- **teo Wagner–Whitin**: ∃ una sol ottima in cui non si ordina mai quando l'inventario è positivo $I_t x_t = 0$
- **conseguenza**: ogni ordine copre esattamente uno o più periodi futuri consecutivi, oppure non viene effettuato

Bilancio globale dei flussi. Riformulazione come problema di cammino minimo su rete compatta

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

➡ ***algo ha compl pol***

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

CAP 8

IMPLEMENTAZIONE DP

La **DP** è uno strumento per risolvere problemi di allocazione discreti, continui e stocastici, usando la ***V()*** come oggetto computaz e evidenziando il ruolo della *variabile di stato* e delle *decisioni di controllo*.

Obiettivi

- da *problemi statici* a **processi sequenziali**
- costruire e approssimare la **value function**
- implementare ***ricorsioni di Bellman*** numeriche
- gestire stati discreti, continui e stocastici
- comprendere *limiti computazionali della DP*

SCORTE STOCASTICHE

Variazione stocastica del problema di lot-sizing. Domanda aleatoria discreta e assunzione di vendite perse. La **Vk(s) è tabulata** e lo stato evolve secondo una ***dinamica con troncamento a zero dell'inventario fisico***. $I_{t+1} = \max\{0,\ I_t + x_t - d_{t+1}\}$

Spazio degli stati e costi. Lo stato è l'inventario ***It***, le azioni sono gli **ordini ammissibili**. Il costo immediato include costo lineare d'ordine e penalità sull'inventario contabile, potenzialmente non simmetrica. $\beta\,(I_t + x_t - d_{t+1})^2$

- **inventario fisico ≥ 0**
- **inventario contabile anche < 0**
- **costo immediato** in f dalla *domanda futura*
- *ricorsione DP con termine di costo stocastico in aspettativa*

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

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

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

POLITICHE S E (s, S)

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

Struttura stocastica del lot-sizing. La Wagner–Whitin non vale, ma sotto hp di *convessità emergono risultati strutturali*. Con **backlog ammesso** e **penalità convessa** ⇒ $V_t(I_t) = \min_{x_t \ge 0} \{ c\,x_t + H(I_t + x_t) + \mathbb{E}[V_{t+1}(I_t + x_t - d_{t+1})] \}$

Minimizzatori e politiche ottime. Il prob può essere analizzato tramite *minimizzatori vincolati e non vincolati* della *f costo attesa*, distinguendo tra livelli ob accessibili e soluzioni di bordo. $G_t(y_t) = c\,y_t + H(y_t) + \mathbb{E}[V_{t+1}(y_t - d_{t+1})]$

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

- **politica base-stock**: si ordina fino al livello obiettivo S quando l'inventario è sotto S
- **politica (s, S)**: si ordina solo se *It* < *s*, riportando l'inventario a *S* ⇒ *ottim in stazionario*
- **limiti della DP**: dimensionalità dello stato, ottimizzazione, aspettative, modellazione

CAP 9
IMPLEMENTAZIONE DP

Principio di modellazione per problemi decisionali multistadio, su **decomposizione in sottoproblemi** a **singolo stadio** e sull'uso della **V()** per bilanciare contributi immediati e futuri, considerando *azioni ammissibili*, *incertezza* e *dinamica dello stato*.

Obiettivi

- implementare *ricorsioni di Bellman*
- gestire **stati discreti** e **dinamiche stocastiche**
- comprendere i *limiti computazionali* e di *modellazione della DP*

VALUTAZIONE POLITICHE

Limiti degli MDP classici. Nei **MDP finiti** la valutazione diretta tramite V() è concettualmente semplice, ma nella pratica è spesso ostacolata dalla **curse of modeling** e dalla difficoltà di specificare o stimare le **probabilità di transizione**.

Per aggirare tali limiti si ricorre:

- al ***campionamento Monte Carlo***
- a ***riformulazioni alternative*** della ricorsione

Q-factors e policy iteration. I **Q-factors** *misurano il valore di intraprendere una certa azione in uno stato dato*, assumendo una politica futura fissata. Essi sono centrali nei metodi di *policy evaluation (se seguo la politica quanto è buona) e policy iteration (miglioramento della policy)*, consentendo di migliorare iterativamente una politica candidata.

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

- definizione di **Q(i,a)** come *valore stato-azione*
- relazione tra **Q-factors** e **V()**

$$Q(i,a) = f(i,a) + \gamma \sum_{j\in S} \pi(i,a,j)\,\left[\mathrm{opt}_{\tilde{a}\in A(j)}\, Q(j,\tilde{a})\right]$$

Vantaggi e svantaggi. L'uso dei ***Q-factors aumenta la dimensionalità*** del problema, ma consente uno scambio tra attesa e ottimizzazione che semplifica il calcolo e abilita *approcci model-free*.

- apprendimento tramite *campionamento*
- utilizzo di **architetture di approssimazione**

Stato post-decisione e modellazione avanzata. L'introduzione dello stato post-decisione separa decisione e incertezza, rendendo l'ottimizzazione deterministica e l'attesa esterna.

$$V_t(s_t) = \mathbb{E}_t\left[\mathrm{opt}_{x_t\in X(s_t)}\,\{f_t(x_t,s_t) + \gamma V_{t+1}(s_{t+1})\}\right]$$

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

$$V_{t-1}^x(s_{t-1}^x) = \mathbb{E}\left[V_t(s_t)\;\middle|\; s_{t-1}^x\right] = \mathbb{E}\left[\mathrm{opt}_{x_t\in X(s_t)}\,(f_t(s_t,x_t) + \gamma V_t^x(s_t^x))\;\middle|\; s_{t-1}^x\right]$$

- inventario **on-hand** e **on-order**
- gestione di **lead time** e pipeline di consegna

PRINCIPIO DI MODELLAZIONE DP

Idea. Il *principio DP* è un concetto per **decomporre un problema decisionale multistadio in una sequenza di problemi a singolo stadio**, bilanciando *contributi immediati e contributi attesi futuri* tramite V().

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

Riformulazioni. In alcuni casi si scambiano attesa e ottimizzazione introducendo **Q-factors** (rappresenta il valore delle coppie stato-azione) e **stato post-decisione**, semplificando la struttura computazionale.

- separazione tra *decisione* e *incertezza*
- supporto a problemi di grande scala

Processi decisionali di Markov. Nei **MDP** finiti stati e azioni sono discreti e la ***dinamica è descritta da probabilità di transizione dipendenti dall'azione***.

- *formulazione tabellare* delle transizioni
- *ricorsioni DP esplicite*

1. ***finito*** *dim* $V_t(i) = \mathrm{opt}_{a\in A(i)}\left\{f_t(i,a) + \gamma \sum_{j\in S} \pi_{t+1}(i,a,j)\, V_{t+1}(j)\right\}$
2. ***infinito*** *scontato* $V(i) = \mathrm{opt}_{a\in A(i)}\left\{f(i,a) + \gamma \sum_{j\in S} \pi(i,a,j)\, V(j)\right\}$
3. *contrib immed stocastico*

$$V(i) = \mathrm{opt}_{a\in A(i)} \sum_{j\in S} \pi(i,a,j)\;\{h(i,a,j) + \gamma V(j)\}$$

REVENUE MANAGEMENT

Idea. Il revenue management raccoglie **modelli e tecniche per massimizzare il ricavo dalla vendita di risorse deperibili**, controllando la disponibilità anziché i prezzi. Le decisioni dipendono dal **trade-off** tra *quantità allocate alle classi* e *qualità dei clienti*, dal *comportamento di acquisto* (***mercato segmentato***) e dal *timing della domanda* (***domanda sequenziale***).

Approcci e struttura dei modelli.

- **quantity-based**: controllo tramite limiti di capacità per classi a prezzi decrescenti, con costo marginale nullo.
- **price-based**: controllo tramite scelta dei prezzi, influenzando direttamente la domanda.

1. **Modello statico con segmentazione perfetta**. Le domande di classe sono indipendenti e sequenziali. La ***decisione ottima non dipende dalla distribuzione di Dj*** e la *ricorsione DP assume forma scambiata*, basata sul valore marginale atteso della capacità.

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

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

- *monotonicità del valore della capacità* $\Delta V_j(s+1) \le \Delta V_j(s)$ e $\Delta V_{j+1}(s) \ge \Delta V_j(s)$
- *livelli di protezione annidati* $y_j^* := \max\{y :\; p_{j+1} < \Delta V_j(y)\}$

2.**Modelli dinamici.**

2a. Segmentazione perfetta la **politica ottima dipende dal tempo**.

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

2b. Scelta del cliente, la *decisione riguarda l'insieme di classi offerte e l'ottimizzazione resta nella forma standard di valore atteso.*

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

CAP 10
DP NUMERICA PER STATI DISCRETI

La **DP per stati discreti** studia problemi decisionali multistadio in cui lo stato evolve a tempo discreto su uno spazio finito o numerabile secondo una **dinamica markoviana**, controllata tramite azioni, e le prestazioni sono valutate nel breve o nel lungo periodo tramite funzioni di valore.

Obiettivi

- modellare con **Markov Decision Processes**
- definire e interpretare la **V()** e le **politiche**
- risolvere **problemi a orizzonte finito e infinito**
- ***algo numerici** di **value iteration** e **policy iteration***
- il legame con metodi di ***reinforcement learning***

MDP PER STATI DISCRETI

Catene di Markov a tempo discreto. Processi stocastici su spazio di stati discreto in cui la dinamica dipende solo dallo *stato corrente*. Nei **Markov Decision Processes** la transizione è parzialmente controllata tramite azioni, introducendo una *struttura decisionale sulla catena*.

MDP a orizzonte finito e infinito. A orizzonte finito la V() dipende dal tempo e confronta ricompensa immediata e valore scontato dell'attesa.

A orizzonte infinito V() e è soluzione di un'equazione ricorsiva di punto fisso, indipendente dal tempo.

$$V_t(i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \left\{ f(i,a) + \gamma \sum_j \pi(i,a,j)\, V_{t+1}(j) \right\}$$

- ***ricorsioni di Bellman*** con dipendenza temporale

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

- confronto *stop vs wait* negli *es di arresto ottimo*
- **equazioni DP come problema di punto fisso**

1. **Value iteration** $[\mathcal{T}\tilde{V}](i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \sum_{j\in\mathcal{S}} \pi(i,a,j)\{h(i,a,j)+\gamma\tilde{V}(j)\}$
2. **Policy iteration** $[\mathcal{T}_\mu\tilde{V}](i) = \sum_{j\in\mathcal{S}} \pi(i,\mu(i),j)\{h(i,\mu(i),j)+\gamma\tilde{V}(j)\}$

POLICY ITERATION

Definizione e policy improvement. La ***policy iteration*** è un metodo iterativo che alterna la **valutazione esatta della V() associata a una politica stazionaria fissata e un passo di miglioramento della politica**, ottenuto scegliendo in ogni stato l'azione che massimizza il valore atteso.

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

Il **sistema di eq lineari** è

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

Valutazione e confronto dei metodi. La valutazione della politica richiede la *soluzione di un sistema lineare*, mentre il *miglioramento* garantisce che la *nuova politica non peggiori la precedente*. Rispetto alla value iteration, ***la policy iteration converge in un numero finito di passi, ma con iterazioni più costose***.

$$\tilde{\mu}(i) \in \arg\ \underset{a\in\mathcal{A}(i)}{\text{opt}} \left\{ f(i,a) + \gamma \sum_{j\in\mathcal{S}} \pi(i,a,j) V_\mu(j) \right\}$$

- **optimistic policy iteration**: variante in cui la policy evaluation è incompleta, con *poche iterazioni prima del miglioramento*
- **generalized policy iteration**: *valutazione e miglioramento della policy procedono simultaneamente*

$$V^{(k+1)}(i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \sum_{j\in\mathcal{S}} \pi(i,a,j)\big\{h(i,a,j)+\gamma V^{(k)}(j)\big\}$$

Algorithm 3 Policy iteration

1: Definire una politica stazionaria iniziale arbitraria $\mu^{(0)}$.
2: Porre $k = 0$ e `stop = false`.
3: **while** `stop` $\neq$ `true` **do**
4: Valutare la politica $\mu^{(k)}$ risolvendo

$$(\mathbf{I} - \gamma\Pi_{\mu^{(k)}})\, V_{\mu^{(k)}} = f_{\mu^{(k)}}.$$

5: Trovare una nuova politica stazionaria $\mu^{(k+1)}$ tramite *policy improvement*

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

6: **if** $\mu^{(k+1)} = \mu^{(k)}$ **then**
7: `stop = true`.
8: **else**
9: $k \leftarrow k+1$.
10: **end if**
11: **end while**
12: Restituire $V_{\mu^{(k)}}$ e $\mu^{(k)}$.

VALUE ITERATION

MDP finiti con sconto stretto. Con fattore di sconto $\gamma<1$, la ***value iteration*** è un ***metodo numerico che calcola la funzione di valore ottima come punto fisso dell'operatore di Bellman***.

$$V^{(1)}(i) = [\mathcal{T}V^{(0)}](i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \sum_{j\in\mathcal{S}} \pi(i,a,j)\big\{h(i,a,j)+\gamma V^{(0)}(j)\big\}$$

$$V^{(k+1)}(i) = [\mathcal{T}V^{(k)}](i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \sum_{j\in\mathcal{S}} \pi(i,a,j)\big\{h(i,a,j)+\gamma V^{(k)}(j)\big\}$$

Definizione e operatore H. La V() consiste nell'applicare iterativamente l'operatore H a una V() iniziale fino a convergenza. *H combina ricompense immediate e valore futuro scontato*, producendo una successione di stime sempre più accurate.

$$H(\cdot) : \mathbb{R}^n \to \mathbb{R}^n \quad \mathbf{y}^{(k+1)} = H(\mathbf{y}^{(k)})$$

➡️***aggiornamento iterativo*** di V() tramite *applicazioni successive di H*

➡️**arresto tramite criterio di tolleranza** sulla norma della differenza tra iterazioni successive

Algorithm 2 Value iteration (MDP finito con sconto stretto)

1: Scegliere una funzione di valore iniziale $V^{(0)}$; se non vi sono indicazioni, porre $V^{(0)}(i) = 0$ per ogni $i \in \mathcal{S}$.
2: Scegliere un parametro di tolleranza ϵ.
3: Porre $k = 0$ e `stop = false`.
4: **while** `stop` $\neq$ `true` **do**
5: **for all** $i \in \mathcal{S}$ **do**
6: Calcolare

$$V^{(k+1)}(i) = \underset{a\in\mathcal{A}(i)}{\text{opt}} \left\{ f(i,a) + \gamma \sum_{j\in\mathcal{S}} \pi(i,a,j) V^{(k)}(j) \right\}.$$

7: **end for**
8: **if** $\|V^{(k+1)} - V^{(k)}\|_\infty < \epsilon$ **then**
9: `stop = true`.
10: **else**
11: $k = k+1$.
12: **end if**
13: **end while**
14: Porre $\tilde{V} = V^{(k+1)}$.
15: Trovare la politica ottima stimata scegliendo un'azione arbitraria se l'insieme delle azioni ottime non è un singleton

$$\hat{\mu}(i) \in \arg\ \underset{a\in\mathcal{A}(i)}{\text{opt}} \left\{ f(i,a) + \gamma \sum_{j\in\mathcal{S}} \pi(i,a,j) \tilde{V}(j) \right\}, \quad i \in \mathcal{S}.$$

16: Restituire $\tilde{V}$ e $\hat{\mu}$.

CAP 11
PROGRAM DP STATI DISCRETI

La **DP approssimata estende la DP classica** a MDP discreti di grandi dimensioni o a orizzonte infinito, *rinunciando all'ott esat* per **superare le maledizioni computaz**. Nel caso *model-free*, questi metodi coincidono con il ***RL*** e si basano sull'apprendimento tramite interazione e campionamento.

Obiettivi

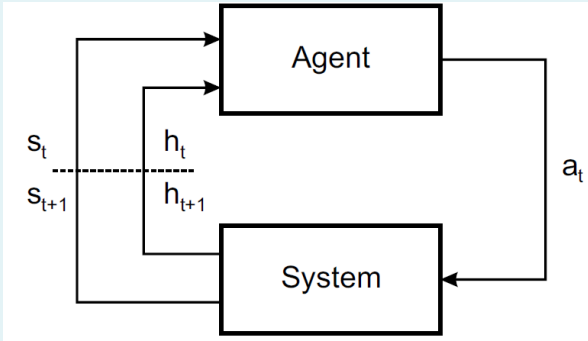
- affrontare ***MDP*** *discreti senza modello esplicito*
- **approssimare** value function e Q-factor
- gestire **exploration** vs **exploitation**
- applicare **SARSA** e **Q-learning**

ADP

ADP. La ***approximate dynamic programming*** raccoglie metodi DP che rinunciano alla garanzia di ottimalità per ridurre la complessità comp e rendere *trattabili MDP discreti a orizzonte infinito*.

RL. Il ***reinforcement learning*** comprende versioni *model-free* di value iteration e policy iteration, in cui un *agente apprende una politica di controllo interagendo con il sistema*.

- **bilanciamento** tra obb di breve e lungo periodo tramite *reward e V()*



SMOOTHING ESPONENZIALE

Non stazionarietà. Anche con dinamica stazionaria, la politica evolve durante l'apprendimento, rendendo non stazionario il bersaglio stimato. ***La media campionaria assegna pesi uniformi e diventa poco reattiva***.

$$\hat{\theta}^{(m)} = \frac{1}{m}\sum_{k=1}^{m} X^{(k)} = \frac{1}{m}\left(X^{(m)} + \sum_{k=1}^{m-1} X^{(k)}\right) = \frac{1}{m}\left(X^{(m)} + (m-1)\hat{\theta}^{(m-1)}\right)$$

$$= \frac{1}{m}X^{(m)} + \frac{m-1}{m}\hat{\theta}^{(m-1)} = \hat{\theta}^{(m-1)} + \frac{1}{m}\left(X^{(m)} - \hat{\theta}^{(m-1)}\right).$$

Smooting esponenziale. Lo SE aggiorna la stima con correzioni proporzionali all'errore corrente.

- **learning rate α** costante o decrescente
- **compromesso** tra reattività e stabilità

$$\hat{\theta}^{(m)} = \alpha X^{(m)} + (1-\alpha)\hat{\theta}^{(m-1)} = \alpha X^{(m)} + \alpha(1-\alpha)X^{(m-1)} + (1-\alpha)^2\hat{\theta}^{(m-2)}$$

$$= \sum_{k=0}^{m-1} \alpha(1-\alpha)^k X^{(m-k)} + (1-\alpha)^m \hat{\theta}^{(0)},$$

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty \quad \text{e} \quad \sum_{k=1}^{\infty} \left[\alpha^{(k)}\right]^2 < \infty$$

SARSA

Operatore e punto fisso. Il valore $V\mu$ associato a una politica μ è il punto fisso dell'operatore $T\mu$. $\mathcal{T}_\mu V_\mu = V_\mu$

$$[\mathcal{T}_\mu \tilde{V}](i) = \sum_{j\in S} \pi(i,\mu(i),j)\big(h(i,\mu(i),j) + \gamma \tilde{V}(j)\big)$$

In modo *equivalente*, i *Q-factor della politica soddisfano un'equazione di punto fisso basata su contributo immediato e valore futuro*.

$$Q_\mu(i,\mu(i)) = \mathbb{E}[h(i,\mu(i),j) + \gamma Q_\mu(j,\mu(j))]$$

Difficoltà comp. La risoluzione diretta dell'equazione di punto fisso non è praticabile: le *probabilità di transizione sono ignote* e *l'iterazione diretta non garantisce convergenza*.

SARSA. *SARSA apprende i Q-factor usando* ***temporal differences***, cioè l'errore tra reward osservato più valore futuro stimato e la stima corrente. $\Delta^{(k)} = h(i,\mu(i),j) + \gamma \hat{Q}_\mu^{(k-1)}(j,\mu(j)) - \hat{Q}_\mu^{(k-1)}(i,\mu(i))$

$$\hat{Q}_\mu^{(k)}(i,\mu(i)) = \hat{Q}_\mu^{(k-1)}(i,\mu(i)) + \alpha\,\Delta^{(k)}$$

- **metodo di bootstrapping**: *usa stime per aggiornare stime*
- **on-policy**: *l'aggiornamento utilizza l'azione scelta dalla politica corrente*
- integrazione con **exploration** e *generalized policy iteration* $\tilde{\mu}(i) \in \arg\underset{a\in A(i)}{\text{opt}}\ \hat{Q}_\mu(i,a)$

EXPLORATION – EXPLOITATION

Campionamento e stima in contesti non stazionari. Nei **MDP** *l'ob non è stimare una quantità statica via Monte Carlo, ma i valori di stato V(i) o i Q-factor Q(i,a).* L'apprendimento deve quindi gestire *exploration-exploitation* e un bersaglio che evolve con la politica.

$$\theta = \mathbb{E}[h(X)] = \int_{\mathcal{X}} h(x) f_X(x)\,dx \qquad \hat{\theta} = \frac{1}{m}\sum_{k=1}^{m} h(X^{(k)})$$

Def.

1. **Exploration** esplora azioni poco conosciute
2. **Exploitation** sfrutta le stime correnti

- **greedy**: seleziona sempre l'azione con valore stimato massimo, senza exploration
- **ε-greedy statico e dinamico**: sceglie l'azione migliore con probabilità 1–ε ed esplora con probabilità ε, ridotta nel tempo nella versione dinamica $\varepsilon^{(k)} = \frac{c}{d+k}$ $\varepsilon^{(k)} = d + \frac{c}{k}$
- **Boltzmann exploration (soft-max)**: assegna probabilità alle azioni in base ai valori stimati, controllando l'exploration tramite un parametro $\varepsilon(a) = \frac{\exp(\rho\,\hat{\nu}(a))}{\sum_{a'\in A}\exp(\rho\,\hat{\nu}(a'))}$

Q-LEARNING

Def. Il **Q-learning** è un metodo di RL model-free che apprende direttamente i Q-factor ottimi, combinando contributo immediato e valore ottimo stimato dello stato successivo tramite smoothing esponenziale. La *politica è implicita nelle stime* e *cambia durante l'apprendimento*. $Q(i,a) = \sum_{j\in S} \pi(i,a,j)\Big(h(i,a,j) + \gamma \underset{a'\in A(j)}{\text{opt}}\ Q(j,a')\Big)$

Differenze con SARSA. A differenza di SARSA, *il Q-learning è off-policy*: *utilizza l'azione ottima stimata nello stato successivo, indipendentemente dall'azione effettivamente eseguita.*

- aggiornamento tramite **temporal differences**
- **logica off-policy** $\hat{Q}^{(k)}(s^{(k)},a^{(k)}) = \alpha\hat{q} + (1-\alpha)\hat{Q}^{(k-1)}(s^{(k)},a^{(k)})$
- **convergenza legata a exploration e learning rate**

$$\Delta^{(k)} = \Big[h(s^{(k)},a^{(k)},j) + \gamma \underset{a'\in A(j)}{\text{opt}}\ \hat{Q}^{(k-1)}(j,a')\Big] - \hat{Q}^{(k-1)}(s^{(k)},a^{(k)})$$

$$\hat{Q}^{(k)}(s,a) = \begin{cases} \hat{Q}^{(k-1)}(s,a) + \alpha\,\Delta^{(k)}, & \text{se } s = s^{(k)},\ a = a^{(k)}, \\ \hat{Q}^{(k-1)}(s,a), & \text{altrimenti,} \end{cases} \qquad V(i) = \underset{a\in A(i)}{\text{opt}}\ Q(i,a).$$

CAP 12
MATLAB

MANCA

MANCA

MANCA

MCNA

- **MANCA**

MANCA

MANCA

MANCA

MANCA

MANCA –

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA

MANCA