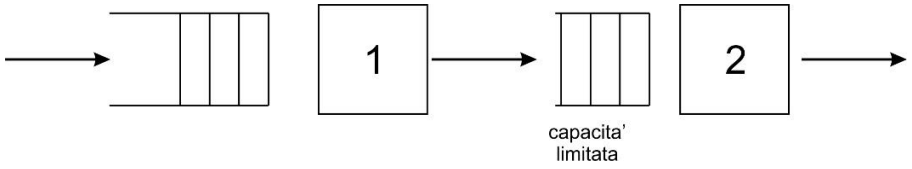
ESAME BA 25/06/2025 **CLOSED BOOK – 90min**

Domanda 1



La figura schematizza un sistema di produzione con due macchine in serie. Il buffer tra le due macchine ha capacità limitata, per cui quando esso è pieno, si può avere blocking sulla macchina 1 (quando ha finito un pezzo, ma non lo può scaricare nel buffer a valle). La macchina 1 viene sbloccata, e quindi può riprendere a lavorare, quando si libera un posto nel buffer. Le code sono gestite FIFO.
Supponiamo che l'arrivo dei pezzi alla macchina 1 segua un processo di Poisson con tasso dato, e che i tempi di lavorazione sulle due macchine seguano due distribuzioni uniformi, con limiti noti.
Scrivere un simulatore MATLAB (o Python) che permetta di valutare la percentuale di tempo in cui la macchina 1 è bloccata.

Soluzione
Prima di tutto occorre individuare gli eventi, che sono l'arrivo di pezzo ed il completamento di un pezzo su ciascuna delle due macchine. La lista dei tempi del prossimo evento per ognuno dei tre eventi fa parte dello stato, come pure il clock corrente.

Lo stato include anche la condizione della macchina 1 (idle, busy, blocked), della macchina 2 (idle, busy), il numero di pezzi nei due buffer (supponiamo di non contare l'eventuale pezzo in lavorazione). Per raccogliere la statistica richiesta, occorre anche cumulare i periodi di tempo di blocking e l'eventuale inizio della fase corrente di blocking.

Si definisce poi la gestione di ogni evento.

- Arrivo:
  - schedula prossimo arrivo;
  - se la macchina 1 è idle, schedula Completamento1, altrimenti aggiungi pezzo in coda.
- Completamento1:
  - (gestisci M1) se buffer a valle pieno, blocca M1 e deschedula evento; altrimenti, se ci sono pezzi in attesa su prima coda, passa al prossimo; altrimenti ferma M1 (idle)
  - (gestisci M2) se macchina 2 libera schedula Completamento2; altrimenti incrementa buffer;
- Completamento2:
  - se buffer vuoto, deschedula evento e M2 entra stato idle;
  - altrimenti decrementa buffer e schedula Completamento2; se macchina 1 in blocking sbloccala, registra la lunghezza del periodo di blocking e gestisci M1 (se c'è coda passa al prossimo, altrimenti idle).

Segue **pseudocodice** MATLAB (si può fare uno script, oppure una funzione; in ogni caso, è consigliabile associare una funzione, magari annidata, a ogni evento).

```
% Definisco dati: arrivalRate, lowerProc1, upperProc1, lowerProc2, ...
% upperProc2, bufferSize

% Definisco (e inizializzo) variabili di stato:
% clock, eventTimes (array per tre tipi eventi)
% idle1, blocked1, idle2 (assumo variabili booleane;
% ootrei usare variabili intere con codici di stato predefiniti)
% queueLength, bufferLength, le due lunghezze di coda
% startBlocked, totalBlocked per mercare inizio blocking e cumulare i tempi

% horizon e' l'orizzonte simulato

eventTimes = [exprnd(1/arrivalRate); inf; inf];
clock = 0;
while clock <= horizon % loop di simulazione
    [clock, idx] = min(eventTimes);
    switch  idx
        case 1
            manageArrival();
        case 2
```

```
            manageCompletion1();
        case 3
            manageCompletion2();
    end
end
% gestisco caso in cui la simulazione finisce in stato di blocking
if blocked1
    totalBlocked = totalBlocked + clock -startBlocked;
end
fractionBlocked = totalBlocked /clock;

%%%%%%%%%%%%%%%%%
function manageArrival()
    eventTimes(1) = clock + exprnd(1/arrivalRate);
    if idle1 && (~blocked1) % macchina 1 libera (ma non bloccata)
        idle1 = false;
        eventTimes(2) = clock + unifrnd(lowerProc1,upperProc1);
    else
        queueLength = queueLength + 1;
    end
end

%%%%%%%%%%%%%%%%%
function manageCompletion1()
    % gestisco M1
    if bufferLength == bufferSize  % buffer pieno, blocco
        idle1 = true;
        eventTimes(2) = inf;
        blocked1= true;
        startBlocked = clock;
        return; % non ho nulla da fare, esco
     else
        if queueLength > 0 % se coda, M1 passa al prossimo
            queueLength = queueLength - 1;
            eventTimes(2) = clock + unifrnd(lowerProc1,upperProc1);
        else % coda vuota, M1 idle
            idle1 = true;
            eventTimes(2) = inf;
        end
    end
    % gestisco M2 (non mi preoccupo di blocking)
    if idle2 % macchina 2 ferma, quindi buffer vuoto, parte 2
        idle2 = false;
        eventTimes(3) = clock + unifrnd(lowerProc2,upperProc2);
    else % 2 busy, metto pezzo in buffer
        bufferLength = bufferLength + 1;
    end
end

%%%%%%%%%%%%%%%%%
function manageCompletion2()
    % gestisco M2
    if bufferLength == 0 % buffer vuoto, M2 idle
        idle2 = true;
        eventTimes(3) = inf;
    else
        eventTimes(3) = clock + unifrnd(lowerProc2,upperProc2);
        bufferLenght = bufferLenght - 1;
        if blocked1 % eventuale sblocco di M1
            blocked1 = false;
            totalBlocked = totalBlocked + clock - startBlocked;
            if queueLength > 0 % M1 riparte
                queueLength = queueLength - 1;
                idle1 = false;
                eventTimes(2) = clock + unifrnd(lowerProc1,upperProc1);
```

```
            end
        end
    end
end
```

Domanda 2

Consideriamo la competizione tra due imprese rispetto alle quantità, usando l'equilibrio di von Stackelberg (gioco sequenziale, in cui l'impresa 1 è leader, la 2 follower). I costi di produzione sono, rispettivamente, 3 e 2 per le due imprese. La funzione di domanda inversa è $p = 3000 - 2Q$.
Calcolare le due quantità di equilibrio (giustificando brevemente i passaggi).

Soluzione

Ricaviamo la funzione di reazione del follower, il cui profitto è:

$$[3000 - 2(q_1 + q_2)]q_2 - 2q_2$$

Ponendo a zero la derivata rispetto a q_2, otteniamo

$$R_2(q_1) = \frac{2998}{4} - \frac{q_1}{2}$$

Il profitto del leader è

$$\left[3000 - 2\left(q_1 + \frac{2998}{4} - \frac{q_1}{2}\right)\right]q_1 - 3q_1$$

la cui derivata prima è

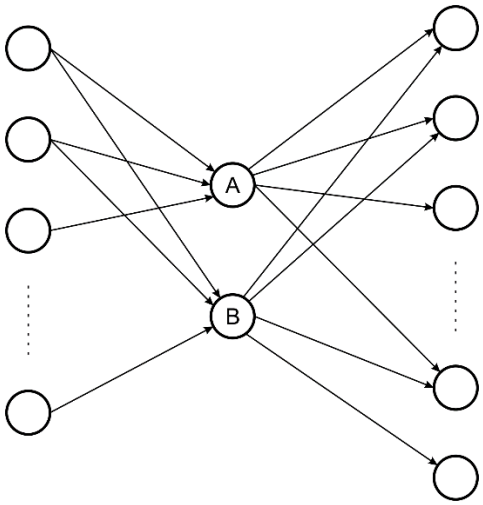
$$\left[3000 - \frac{2998}{2} - 3\right] - 2q_1$$

Si ottiene quindi

$$q_1 = \frac{2997 - 1499}{2} = 749$$

$$q_2 = \frac{2998}{4} - \frac{749}{2} = 375$$

Domanda 3



La rete schematizzata in figura rappresenta una rete di distribuzione con due centri di distribuzione (transit point) A e B. I transit point non servono per immagazzinare, ma per smistare merce dai produttori ai negozi di una grossa catena di vendita. Per semplicità, assumiamo che ogni produttore produca un solo tipo di prodotto, con una capacità data (massimo ammontare disponibile per il prodotto di competenza).
Sempre per semplicità, assumiamo costi di trasporto lineari per ogni arco della rete, con un costo legato al peso complessivo trasportato. Conosciamo il peso unitario di ogni tipo di prodotto.

La domanda per ogni prodotto in ogni punto vendita è incerta e modellata da un insieme di scenari su un solo periodo (quindi non si considerano magazzini). Per ogni prodotto si ha il prezzo di vendita, e si vorrebbe massimizzare il profitto atteso, tenendo conto del ricavo, dei costi di trasporto, e del costo di espansione dei centri di distribuzione A e B (vedere dopo). Non è necessario (e forse nemmeno possibile) soddisfare completamente la domanda.

Si desidera valutare l'opportunità di espandere la capacità di smistamento in A e B, che assumiamo limiti la quantità complessiva di prodotti che può attraversare ciascun singolo transit point (non ci sono magazzini, quindi si ha

conservazione dei flussi). Al momento abbiamo una certa capacità di smistamento, e possiamo espandere uno (uno solo) dei due centri, con un costo noto di espansione e un incremento associato di capacità.

Costruire un modello di programmazione stocastica a due stadi per massimizzare il profitto atteso.

Soluzione
Introduciamo le variabili decisionali binarie di espansione δ_A e δ_B, al primo stadio, a cui associamo i costi di espansione c_A^e e c_B^e. Avremo il vincolo

$$\delta_A + \delta_B \leq 1$$

e un contributo (primo stadio) al profitto atteso $-(c_A^e \delta_A + c_B^e \delta_B)$.

Assumiamo che ci sia corrispondenza 1-1 tra prodotti e produttori (se assumete 1-*N*, è facile adattare il modello).
Introduciamo quindi le variabili decisionali al secondo stadio x_{ij}^{As} e x_{ij}^{Bs} (reali non negative), che rappresentano in ogni scenario *s* il flusso da *i* a *j* che attraversa ciascuno dei due transit point (NB: sfruttiamo la struttura della rete per associare le variabili ai *percorsi*, non ai singoli archi; non è necessario farlo, ma semplifica tutto; sono possibili e corrette anche formulazioni con due set diversi di variabili).
Il vincolo di capacità è:

$$\sum_j \left(x_{ij}^{As} + x_{ij}^{Bs}\right) \leq R_i \qquad \forall i, s$$

dove R_i è la capacità del produttore *i*.
Il vincolo di domanda è quindi

$$x_{ij}^{As} + x_{ij}^{Bs} \leq d_{ij}^s \qquad \forall i, j, s$$

dove d_{ij}^s è la domanda del prodotto *i* in *j* nello scenario *s*.
Il vincolo di capacità nei due transit point è (assumendo di non dovere moltiplicare i prodotti per qualche coefficiente in modo da renderli omogenei in termini di consumo di capacità di gestione del flusso che attraversa il transit point)

$$\sum_i \sum_j x_{ij}^{As} \leq G_A + g_A^e \delta_A \qquad \forall s$$

$$\sum_i \sum_j x_{ij}^{Bs} \leq G_B + g_B^e \delta_B \qquad \forall s$$

dove G_A e g_A^e sono la capacità corrente e quella espansa per A (analogamente per B).

Il termine al secondo stadio nella funzione obiettivo è il valore atteso di ricavo meno costi di trasporto:

$$\sum_s \pi^s \sum_i \sum_j \left[(p_i - c_{ij}^A) x_{ij}^{As} + (p_i - c_{ij}^B) x_{ij}^{Bs}\right]$$

Dove

- π^s è la probabilità dello scenario *s*;
- p_i è il prezzo di vendita del prodotto *i* (potrebbe essere un margine di contribuzione se tendiamo conto dei costi di produzione);
- c_{ij}^A e c_{ij}^B sono i costi dei percorsi $i \to A \to j$ e $i \to B \to j$ (NB: tengono conto della somma dei costi sui due archi attraversati, moltiplicati per il peso).