POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE

IN INGEGNERIA GESTIONALE

TESI DI LAUREA MAGISTRALE

MODELLIZZAZIONE DI UN SISTEMA PRODUTTIVO, STUDIO DEL NUMERO DI OPERATORI E REINFORCEMENT LEARNING

RELATORI
Proff. Dario Antonelli, Agostino Villa

CANDIDATA
Annette Ventroni

A.A. 2020-2021

Contenuti

Allegati

Figure

Tabelle

Abstract

Il lavoro di questa tesi ha consistito:

-nello studio di un sistema produttivo reale: il sistema è stato descritto nel suo comportamento tecnico, modellizzato mediante codice Matlab, è stato studiato il suo funzionamento. È stato poi studiato e analizzato il sistema e sue configurazioni alternative, - mediante simulazione -, con particolare attenzione e interesse al numero di operatori da impiegare e a come impiegarli in produzione, tenendo sotto osservazione la performance del sistema.

- nello studio e valutazione dell'applicazione al sistema in esame del Reinforcement Learning (apprendimento per rinforzo) per l'allocazione degli operatori: si tratta di uno dei paradigmi del Machine Learning (apprendimento automatico), che punta alla realizzazione di agenti autonomi, capaci di scegliere le migliori azioni da eseguire sulla base della propria esperienza passata.

Introduzione

Questa tesi tratta dello studio di un sistema, realmente esistente, che ha come obiettivo la produzione di un pallet metallico, per il quale si vuole valutare la migliore configurazione da attribuire, in termini di numero di operatori, -rispetto alla situazione attuale di occupazione-, e di processo.

Il sistema produttivo viene modellizzato nell'intero del suo comportamento produttivo, mediante codice informatico Matlab, in modo che ne vengano descritte fedelmente sia la struttura che il comportamento.

La modellizzazione effettuata viene accuratamente descritta, e poi utilizzata per simulare il comportamento del sistema nel suo funzionamento, a seconda di varie configurazioni. Si partirà dalla configurazione in uso nell'azienda per verificare in termini pratici il processo in esame e la produzione, per poi valutare invece configurazioni alternative ed osservarne le differenze, al fine di valutare se vi possa essere una configurazione da preferire rispetto a quella attualmente in uso nell'azienda.

L'altro argomento di cui tratta la tesi è l'apprendimento per rinforzo (o Reinforcement Learning, in inglese), paradigma del più noto Machine Learning (apprendimento automatico), che punta a realizzare agenti autonomi in grado di effettuare delle scelte in base alla situazione in cui si trovano. Nel caso del sistema in esame, si potrebbe utilizzare il RL per decidere come allocare gli operatori alle varie attività da eseguire momento per momento, autonomamente, anziché preventivamente, a seconda di una certa funzione da ottimizzare.

Il meccanismo del Reinforcement Learning servirebbe per sostituire al metodo di allocazione classico che impieghiamo nello studio del sistema per l'allocazione degli operatori ai task in ciascun istante di tempo in cui nel processo sia necessaria una o più allocazioni di operatori, statico, con un metodo dinamico, che tenga in considerazione nell'allocazione, della reale condizione del sistema in dato momento della sua evoluzione, piuttosto che di quella ideale solamente.

I due capitoli iniziali sono dedicati al Reinforcement Learning: il primo contiene la teoria sulla matematica su cui si basa, -il Markov Decision Process (MDP) sostanzialmente-, e il secondo è dedicato al Reinforcement Learning stesso: una spiegazione di cosa si tratta, i concetti matematici su cui si basa, le sue parti e concetti fondamentali, come si può applicare, ed altro. Il secondo capitolo è un approfondimento sul Reinforcement Learning e contiene tutte le nozioni e i discorsi necessari in vista dell'ultimo capitolo, -6-, nel quale si tratterà dell'applicazione del RL al sistema in esame, come metodo decisionale per l'allocazione degli operatori ai task.

Il capitolo terzo è dedicato al sistema che abbiamo in esame: viene spiegato come funziona, vengono esplicitati i dati di impianto e di processo, le ipotesi e regole aziendali alla base del suo funzionamento.

Il capitolo quarto è dedicato al modello del sistema che è stato realizzato, in codice Matlab: viene spiegata la rappresentazione che è stata fatta dei dati del sistema e il modello in sé, costituito da una parte di

definizione delle variabili e da una di codice vero e proprio. Il codice nella sua interezza è contenuto nell'allegato 1 di questa tesi.

Il capitolo quinto è dedicato alle simulazioni. Fatte chiare ipotesi, impostazioni e obiettivi della simulazione, attraverso il codice Matlab del capitolo precedente, effettuate eventualmente le necessarie piccole modifiche per il cambio di configurazione sulla quale base simulare e trovare i risultati e, viene simulato il comportamento del sistema in 5 diverse configurazioni. Infine, vengono confrontati i risultati ottenuti e fatta una valutazione sulla configurazione migliore per l'azienda.

Nell'ultimo capitolo viene spiegato il modello che è stato stilato per ottenere i parametri necessari ad applicare la procedura di RL eventualmente al sistema in esame, la procedura di RL che si vorrebbe effettuare, e i risultati ottenuti, oltre che i limiti della procedura. Questo secondo modello è anch'esso allegato alla tesi, - Allegato 2- , ed ha delle parti in comune con il primo.

Seguono le considerazioni finali.

Capitolo 1: Matematica preliminare

1.1 Processo decisionale di Markov (Markov Decision Process)

1.1.1 Introduzione

In matematica, un processo decisionale di Markov (MDP), è un processo di controllo stocastico a tempo discreto; gli MDP forniscono un framework matematico per la modellizzazione del processo decisionale in situazioni in cui i risultati sono in parte casuale e in parte sotto il controllo decisionale.
Gli MDP, noti almeno dal 1950[1] e parte cardine della cui ricerca risultò nel libro di Ronald Howard, datato 1960, "*Dynamic Programming and Markov Processes*"[2], sono utili per lo studio di una vasta gamma di problemi di ottimizzazione, risolti con la programmazione dinamica e l'apprendimento per rinforzo, e sono utilizzati in una vasta area di discipline in cui il processo di presa di decisione avviene in un intorno dinamico, tra cui la robotica, l'automazione, l'economia, e la produzione industriale (che è l'ambito di cui ci occupiamo in questa tesi).
Il nome di questo tipo di processi riprende quello del matematico Russo Andrey Markov, in quanto si tratta di un'estensione delle catene di Markov, che doverosamente riprendiamo nel breve seguito. La differenza tra i due sta nell'aggiunta negli MDP di azioni, che consentono la scelta, e le ricompense, che consentono di dare invece motivazione. Nel caso in cui per ogni stato esista una sola azione (ad esempio "aspetta"), e la ricompensa sia fissata, cioè sempre la stessa, indifferentemente da stato e azione, il MDP si riduce a catena di Markov.
Se gli spazi degli stati e delle azioni sono finiti, il problema è chiamato MDP finito, e gli MDP finiti sono particolarmente importanti per la teoria dell'apprendimento per rinforzo (*reinforcement learning*).

In linguaggio non matematico, per capire come funziona il processo, diciamo che: ad ogni istante temporale il processo si trova in un qualche stato s, e il decisore può scegliere qualsiasi azione a che risulti disponibile in quello stato. Il processo risponde all'istante temporale successivo muovendosi in modo casuale in un nuovo stato s', e dando una ricompensa corrispondente, $R_a(s, s')$.
La probabilità che il processo evolva nel nuovo stato s' è influenzata dall'azione scelta; specificamente, è data dalla funzione di transizione di stato $\boldsymbol{P}_a(s, s')$. Per ciò, lo stato successivo s' dipende dallo stato corrente s e dalla decisione a che il decisore prende.
Però, dato quanto appena detto, è fondamentale il fatto che, dati a ed s, vi è indipendenza condizionale da tutte le azioni e gli stati precedenti, ossia le transizioni di stato di un MDP soddisfano la proprietà di Markov (vedremo anch'essa nel paragrafo dedicato ai concetti preliminari necessari).

[1] Bellman, R., *A Markovian Decision Process*, in *Journal of Mathematics and Mechanics*, vol.6, 1957.
[2] Howard, Ronald A. (1960). *Dynamic Programming and Markov Processes*. The M.I.T. Press.

1.1.2 Processo Markoviano e proprietà di Markov

Si definisce *processo stocastico markoviano* (o *di Markov*), un processo aleatorio in cui la probabilità di transizione che determina il passaggio a uno stato di sistema dipende solo dallo stato del sistema immediatamente precedente (proprietà di Markov) e non da come si sia giunti a tale stato.[3]
Viceversa, si dice *processo non markoviano* un processo aleatorio per cui non vale la proprietà di Markov. Il processo markoviano prende nome dal matematico russo Andrej Andreevič Markov, il quale per primo ne sviluppò la teoria.
Modelli di tipo markoviano vengono utilizzati nella progettazione di reti di telecomunicazioni: la teoria delle code che ne consegue trova applicazione in molti ambiti, dalla fila agli sportelli ai pacchetti dati in coda in un router.

Un processo di Markov può essere descritto per mezzo dell'enunciazione della *proprietà di Markov*, o *condizione di "assenza di memoria"*, che può essere scritta come:

$$\boldsymbol{P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \dots, X(t_0) = x_0) = P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n)}\ .$$

1.1.3 Catene di Markov

Una *catena di Markov* è un processo di Markov con spazio degli stati discreto, quindi è un processo stocastico che assume valori in uno spazio discreto e che gode della proprietà di Markov.
L'insieme S di spazio degli stati può essere finito o infinito numerabile; nel primo caso si parla di catena di Markov a stati finiti.
Una catena di Markov può essere poi, tempo-continua o tempo-discreta, in base all'insieme T di appartenenza della variabile tempo (continuo o discreto).

Formalmente, una catena di Markov è un processo stocastico Markoviano caratterizzato da un parametro $t_i \in T$, da un insieme S di stati e da una funzione probabilità di transizione $\boldsymbol{P} : S\ x\ S\ x\ T \rightarrow [0,1]$.

Trattandosi di processo Markoviano, $\boldsymbol{P}$ gode della proprietà di Markov, che riportiamo tale e quale a sopra:

$$\boldsymbol{P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \dots, X(t_0) = x_0) = P(X(t_{n+1}) = x_{n+1} | X(t_n) = x_n)}$$

Nel caso di catena di Markov a *tempo discreto,* cioè con l'insieme T discreto, la notazione si semplifica:

$$\boldsymbol{P}(X_{n+1} = x_{n+1} | X_n, X_{n-1}, \dots, X_0) = \boldsymbol{P}(X_{n+1} = x_{n+1} | X_n).$$

Una *catena di Markov omogenea* è un processo markoviano in cui la probabilità di transizione al tempo t_i non dipende dal tempo stesso, ma soltanto dallo stato del sistema al tempo immediatamente

[3] *Markov Chain | Definition of Markov Chain in US English by Oxford Dictionaries*, su *Oxford Dictionaries.*

precedente $S(t_{i-1})$. In altre parole, la probabilità di transizione è indipendente dall'origine dell'asse dei tempi e quindi dipende soltanto dalla distanza tra i due istanti temporali.

Per le catene omogenee vale la condizione: $\boldsymbol{P}(X_{n+1} = x|X_n = y) = \boldsymbol{P}(X_n = x|X_{n-1} = y)$.

Più in generale, si dimostra che in una catena di Markov omogenea la probabilità di transizione da uno stato a un altro in n passi è costante nel tempo: $\boldsymbol{P}(X_{i+n} = x|X_i = y) = \boldsymbol{P}(X_{i-1+n} = x|X_{i-1} = y)$.

I sistemi reali che possono essere modellati con catene di Markov omogenee sono rari: è sufficiente pensare al sistema "condizioni atmosferiche" per capire come la probabilità di transizione da uno stato (per esempio "sole") ad un altro stato (per esempio "pioggia") dipende dalla stagione, per cui non è possibile modellare questo sistema come catena di Markov omogenea. Tuttavia, spesso, restringendo l'analisi del sistema a un determinato intervallo di tempo si può considerare il comportamento omogeneo; in questo caso l'intervallo di tempo potrebbe essere una singola stagione.

Una catena di Markov omogenea a stati finiti in cui l'insieme S degli stati del sistema è finito e ha N elementi può essere rappresentata mediante una *matrice di transizione* $A \in \mathbb{R}^{NxN}$ e un vettore di probabilità iniziale $\widetilde{\pi_0} \in \mathbb{R}^N$.

Gli elementi di A rappresentano le probabilità di transizione tra gli stati della catena: una catena che si trova nello stato i ha probabilità a_{ij} di passare allo stato *j* nel passo immediatamente successivo. In particolare, gli elementi a_{jj} sulla diagonale principale, indicano le probabilità di rimanere nello stesso stato i.

Il vettore $\widetilde{\pi_0}$ definisce le probabilità che inizialmente la catena di Markov si trovi in ciascuno degli N stati.

Una catena di Markov omogenea è univocamente definita dalla coppia (A , $\widetilde{\pi_0}$).

Se al tempo t_0 la distribuzione di probabilità è $\widetilde{\pi_0}$, allora le probabilità $\widetilde{\pi_n}$ che a un tempo t_n il sistema si trovi in ciascuno degli N stati sono date dal vettore così definito:

$(\widetilde{\pi_n})^T = (\widetilde{\pi_0})^T x\ A^n$, dove $(\widetilde{\pi_n})^T$ indica la trasposta del vettore $\widetilde{\pi_n}$.

Dalla definizione assiomatica della probabilità discendono le seguenti proprietà per la matrice A:

- $a_{ij} \geq 0, \forall i,j \in \{1 \dots N\}$;

- $\sum_{j=1}^{N} a_{ij} = 1, \forall i \in \{1 \dots N\}$.

La seconda proprietà equivale a richiedere che la somma degli elementi su ciascuna riga sia uguale a 1.

Per esempio, A e π_0 possono essere i seguenti: $A = \begin{pmatrix} 0.3 & 0.5 & 0.2 \\ 0.1 & 0.8 & 0.1 \\ 0.5 & 0.5 & 0 \end{pmatrix}$, $\pi_0 = \begin{pmatrix} 0.4 \\ 0.3 \\ 0.3 \end{pmatrix}$;

nel caso di una *catena di Markov omogenea a stati discreti* si può adottare la notazione sintetica:

$\boldsymbol{P_{ij}^{(n)}} = \boldsymbol{P}(X_{m+n} = s_j|X_m = s_i) = \boldsymbol{P}(X_n = s_j|X_0 = s_i)$, dove (n) non è un esponente bensì un indice.

Si ha quindi: $(\widetilde{\pi_n})_j = \sum_{i \in S} (\widetilde{\pi_n})_i * (\boldsymbol{P}^{(n)})_{ij}$

Valgono dunque le seguenti proprietà:

- $\boldsymbol{P}_{ij} \geq 0, \forall i,j \in S$;

- $\sum_{j \in S} \boldsymbol{P}_{ij} = 1$.

Il *periodo* di uno stato $s_i \in S$ *di una catena di Markov a stati discreti*, con S finito o infinito numerabile, è definito come il minimo numero di passi temporali affinché vi sia una probabilità diversa da zero di tornare sullo stesso stato, partendo dallo stato $s_i$al tempo t_m.
Formalmente, il periodo $d(s_i)$ è definito come segue:

$d(s_i, t_m) = MCD\{n \geq 1 : \boldsymbol{P}(X_{m+n} = s_i | X_m = s_i) > 0\}$, dove MCD indica il massimo comune divisore.

Nel caso di una catena di Markov omogenea a stati finiti con numero di stati N, rappresentabile quindi con una matrice $A \in \mathbb{R}^{NxN}$, si può riformulare la definizione così:

$d(s_i) = MCD\{n \geq 1 : (\boldsymbol{A^n})_{\boldsymbol{ii}} > 0\}$.

Lo *stato* s_i è detto *aperiodico* se il suo periodo è uguale a 1.

Una *catena di Markov* è detta *aperiodica* se tutti i suoi stati sono aperiodici, altrimenti è detta periodica.

Se una catena di Markov è irriducibile allora tutti i suoi stati hanno lo stesso periodo.

Una *catena di Markov a stati discreti* è detta *irriducibile* se partendo da ogni stato s_i c'è una probabilità maggiore di zero di raggiungere ogni altro stato s_j.
Formalmente, una catena di Markov è irriducibile se:

$\forall\, s_i, s_j \in S, \forall\, m \in \mathbb{N}, \exists\, \boldsymbol{n} \in \mathbb{N} : \boldsymbol{P}(X_{m+n} = s_j | X_m = s_i) > 0$.

1.1.4 Definizione

Un MDP finito è definito da:

- uno spazio degli stati S;
- uno spazio delle azioni $A = U_s \in_s A(s)$ che possono essere intraprese in funzione dello stato;
- le probabilità di transizione $\boldsymbol{P}_a(s, s') : S\ x\ A\ x\ S \to \mathbb{R}$ definiscono le dinamiche one-step dell'ambiente, ovvero, la probabilità che, dato uno stato s e un'azione a al tempo t, si raggiunga il possibile stato successivo s': $\boldsymbol{P}_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$;
- il valore atteso della ricompensa $R_a(s, s')$: dato uno stato s e un'azione a, se si passa allo stato s' si ottiene una ricompensa pari a $R_a(s, s') = R(s'|s, a) = \mathbf{E}(R_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$;
- $\gamma \in [0,1]$ è il fattore di sconto (discount factor) che rappresenta l'importante differenza tra le ricompense future (*future rewards*) e le ricompense presenti (*present rewards*).

(Nota: la teoria del processo decisionale di Markov non specifica che S o A sono finiti, ma gli algoritmi di base assumono che lo siano.)

1.1.5 Scopo

Il problema centrale di un MDP è quello di identificare quale sia la migliore azione da eseguire in un dato stato, in modo da ottenere il massimo valore possibile di una funzione cumulativa della ricompensa.
La funzione che per ogni stato $s \in S$ identifica l'azione $a \in A$ da applicare è chiamata "politica" (*policy*) stazionaria $\pi: S \mapsto A$. Tipicamente, questa funzione che valuta la ricompensa è il valore atteso di una somma con sconto su un orizzonte potenzialmente infinito:

$\sum_{t=0}^{\infty} \gamma^t \cdot R_{a_t}(s_t, s_{t+1})$ dove $a_t = \pi(s_t)$ sono le azioni date dalla politica π e γ è il fattore di sconto compreso fra 0 e 1.

Per via della proprietà di Markov, la politica ottimale $\pi *$ che massimizza la ricompensa con sconto attesa, per questo problema può essere scritta come una funzione del solo stato s. Per ottenere una politica ottimale in tempo polinomiale per un MDP dato, spesso si usano algoritmi di programmazione lineare o, più tradizionalmente, di programmazione dinamica[2].

1.1.6 Algoritmi

La famiglia standard di algoritmi che calcola questa politica ottimale fa uso di due vettori, indicizzati con lo stato, che si chiamano: "valore" V, che contiene valori reali, e "politica"' π, che contiene le azioni. Quando l'algoritmo termina, π contiene la politica soluzione e $V(s)$ contiene la somma con sconto delle ricompense da ottenere (in media) seguendo la soluzione $\pi(s)$ a partire dallo stato s.
L'algoritmo prevede due tipi di passo: un aggiornamento del valore e un aggiornamento della politica, i quali sono ripetuti in tutti gli stati e in un certo ordine, fintanto che non avvengano più cambiamenti ai valori. I due aggiornamenti ricalcolano ricorsivamente un nuovo valore stimato per la politica ottimale e per il valore dello stato, usando una stima precedente di questi valori.

$$\pi(s) := argmax_a \left\{ \sum_{s'} \boldsymbol{P}_a(s, s')(\boldsymbol{R}_a(s, s') + \gamma V(s')) \right\}$$

$$V(s) := \sum_{s'} \boldsymbol{P}_{\pi(s)}(s, s')(\boldsymbol{R}_{\pi(s)}(s, s') + \gamma V(s'))$$

L'ordine degli aggiornamenti dipende dalla variante dell'algoritmo: si possono applicare su tutti gli stati assieme, stato per stato, sequenzialmente, o anche più spesso per certi stati che per altri. Fintanto che nessuno stato viene escluso dai passi di aggiornamento, l'algoritmo eventualmente convergerà verso una soluzione [4].

[4] Stewart N. Ethier, *Markov processes: characterization and convergence,* Wiley, 1986, ISBN 9780470316658, OCLC.

Capitolo 2: L'apprendimento per rinforzo

2.1 Introduzione

L'apprendimento per rinforzo (o Reinforcement Learning) è una tecnica di apprendimento automatico che punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per il conseguimento di determinati obiettivi tramite interazione con l'ambiente in cui sono immersi.

È uno dei tre paradigmi principali dell'apprendimento automatico (*machine learning*), insieme all'apprendimento supervisionato e a quello non supervisionato. A differenza degli altri due, l'apprendimento per rinforzo si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. A differenza dell'apprendimento supervisionato, inoltre, non necessita che siano presentati input/output categorizzati.

La qualità di un'azione è data da un valore numerico di "ricompensa", ispirata al concetto di rinforzo, che ha lo scopo di incoraggiare comportamenti corretti dell'agente.

Fondamentale, nell'apprendimento per rinforzo, è trovare un corretto bilanciamento tra l'esplorazione del problema (non precedentemente esplorato e rappresentato in qualche modo) e lo sfruttamento della conoscenza acquisita; nel mondo anglosassone ci si riferisce a questo obiettivo di bilanciamento con '*exploration vs exploitation*' [5].

Questo tipo di apprendimento è solitamente modellizzato tramite i processi decisionali di Markov e può essere effettuato mediante diverse tipologie di algoritmi, classificabili in base all'utilizzo di un modello che descriva l'ambiente, alle modalità di raccolta dell'esperienza (in prima persona o da parte di terzi), al tipo di rappresentazione degli stati del sistema e delle azioni da compiere (discreti o continui).

L'algoritmo per il RL punta a "costruire" un agente in grado da sé di prendere delle buone decisioni, ma senza che esso sia "preparato" direttamente circa cosa fare; in vista di un suo obiettivo. L'agente imparerà da sé, attraverso l'esperienza diretta e l'interazione con l'ambiente in cui è inserito, che gli rimanderà indietro dei segnali, allo stesso modo in cui un bambino può imparare a fare dei movimenti. Ad esempio, infatti, ad un bambino piccolo non avrebbe senso fare un 'training' specifico ed esplicito a parole, mancandogli la possibilità di comprendere a pieno il significato delle parole, e allora capita che impari senza che qualcuno gli dica direttamente cosa fare, ma piuttosto da sé, provando e riprovando a fare movimenti e vedendo cosa succede.

Dunque, la base del concetto di RL è che vi sia un ambiente, nel quale è immerso un agente decisionale, a cui l'ambiente risulta esterno, che deve fare qualcosa in tale ambiente e quindi deve decidere e poi intraprendere delle azioni (in sequenza). Ciascuna azione lo porta in uno stato successivo, del quale viene "informato" dall'ambiente, e per ognuna delle quali riceve un segnale, sempre da parte dell'ambiente, -

[5] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301. S2CID 1708582. Archived from the original on 2001-11-20.

chiamiamola una “*ricompensa”* -, che può essere più o meno “buona” o “cattiva”, a testimonianza del fatto che abbia più o meno scelto di fare una azione corretta in vista dell’obiettivo.

Questa è l’idea di base, a questo livello molto astratta e un po’ imprecisa in quanto vi sono vari modi di impostare le ricompense e non possiamo, molto semplicemente, dire che <<più è basso il valore di ricompensa legato ad un’azione, più quell’azione è “cattiva”>>: si potrebbe decidere di impostare per certe coppie stato-azione delle ricompense basse per dare più peso a delle ricompense future, ad esempio.

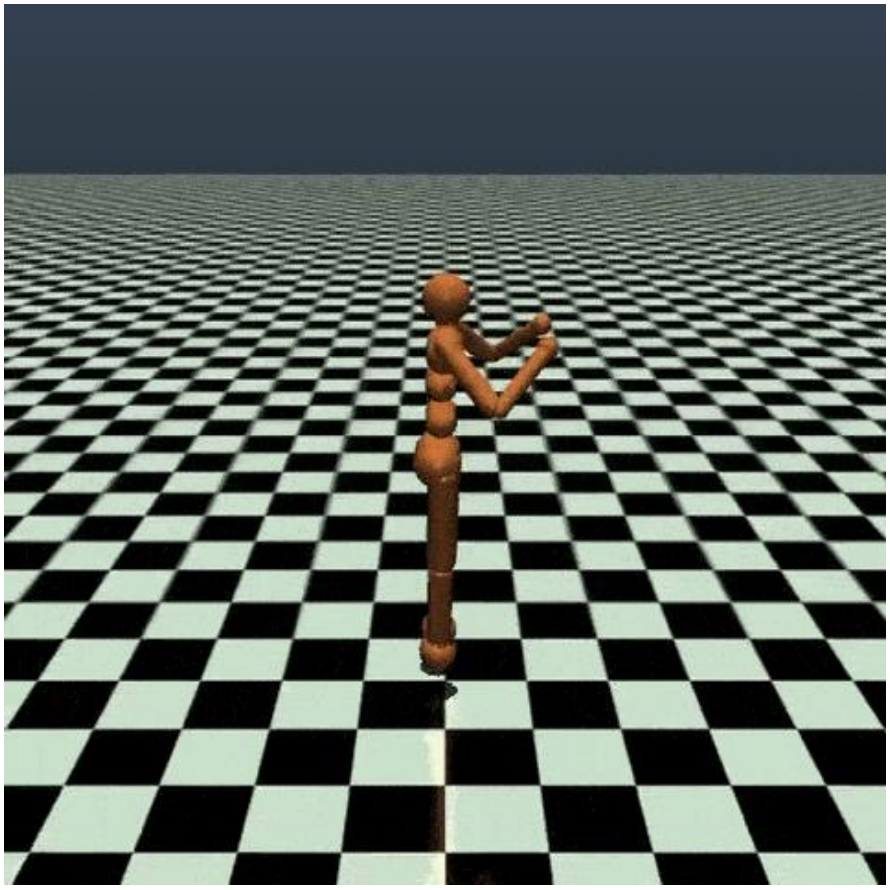


FIGURA 1.1 - ESEMPIO: UMANOIDE IMPARA A CORRERE

In figura 1.1 abbiamo una simulazione, a scopo di migliore comprensione di quanto introdotto, di un umanoide che ha imparato a correre, dopo che ha eseguito la sequenza di azione, osservazione e nuovamente azione, fino a che ha finalmente capito quale fosse la migliore azione da intraprendere in ciascuno step in vista del raggiungimento del suo obiettivo, cioè correre effettivamente.

2.2 Descrizione

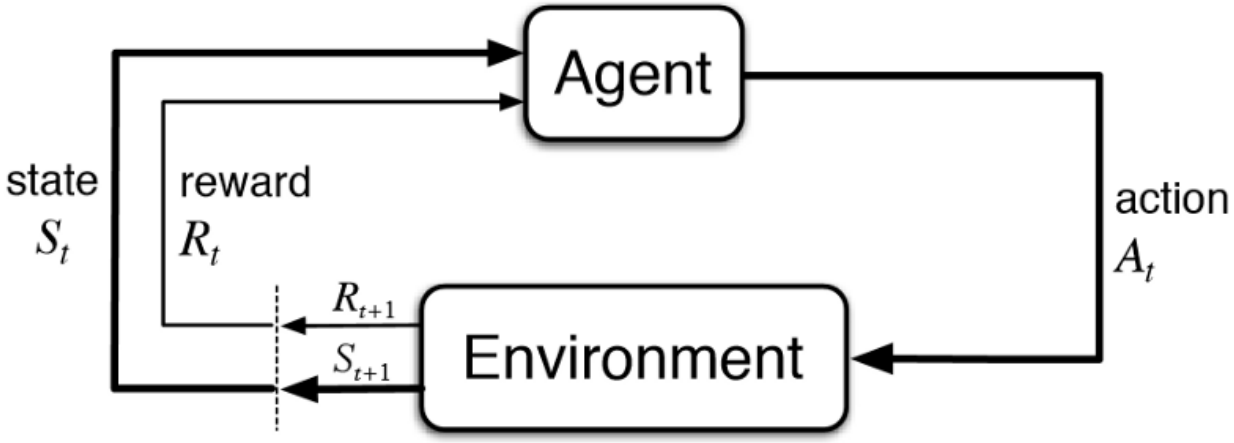


FIGURA 1.2 - SCHEMA DI BASE DEL MECCANISMO DEL RL

In figura 1.2, abbiamo una rappresentazione del meccanismo di base del RL, che procediamo a spiegare. L'agente e l'ambiente interagiscono su una sequenza di istanti temporali, che ipotizziamo discreta (in tutto il proseguo di questa tesi, non verranno considerate sequenze di tempo continue, ma sempre discrete o discretizzabili) $t = 0,1,2, \dots$. Ad ogni istante temporale l'agente riceve una qualche rappresentazione dello stato dell'ambiente, $S_t \in S$, e sulla base di ciò seleziona un'azione $A_t \in A$. All'istante temporale immediatamente successivo, in parte in conseguenza all'azione che ha intrapreso, l'agente riceve una ricompensa numerica $R_{t+1} \in R \subset \mathbb{R}$, e si ritrova in un nuovo stato: $S_{t+1} \in S$.

L'intera sequenza di osservazioni, azioni e ricompense durante l'intera "vita" dell'agente fino all'istante t è detta la *storia*: $H_t = S_1, A_1, R_2, \dots, S_{t-1}, A_{t-1}, R_t$. La storia è ciò che l'agente ha visto, ossia tutte le variabili osservabili, fino all'istante t .

Quello che accade successivamente dipende dalla storia: l'agente seleziona le azioni, per cui vi è una mappatura della storia nell'azione. L'ambiente, come ormai si è ben capito, "emette" un nuovo stato ed una ricompensa associata all'azione scelta.

Innocentemente, sembrerebbe che usare la storia sia il modo migliore per codificare ciò che l'agente ha incontrato sino a quel momento, e che la cosa migliore da fare sarebbe scegliere l'azione successiva in base a ciò; ma la storia non è molto utile dal momento che è enorme, e questo approccio sarebbe infattibile negli interessanti problemi del mondo reale.

Al contrario, si guarda allo stato, che è come un sommario dell'informazione riscontrata fino al momento della decisione; è solo in base allo stato attuale che si decide quale sarà l'azione successiva. Lo stato, comunque, è funzione della storia: $S_t = F(H_t)$.

Lo *stato* è un concetto fondamentale per il RL.

Lo *stato dell'ambiente*, S_t, dà una descrizione dell'ambiente (secondo determinati parametri) e, dal punto di vista dell'ambiente (non dell'agente), determina cosa succede successivamente a seconda dell'azione che l'agente sceglie, in termini di stato successivo e di ricompensa. Esso è la rappresentazione privata

dell'ambiente, vale a dire quali che siano i dati che l'ambiente stesso utilizza per scegliere la ricompensa e lo stato successivo.

S_t solitamente non è visibile all'agente, e anche qualora lo sia, potrebbe contenere delle informazioni per lui irrilevanti.

Lo *stato dell'agente* cattura ciò che è successo all'agente fino a quel momento, cosa sta succedendo, e l'agente usa quella informazione per scegliere l'azione successiva. Si tratta della rappresentazione interna: qualsiasi informazione l'agente utilizzi per la scelta dell'azione. Può essere qualsiasi funzione della storia, come menzionato; l'agente decide quale sarà tale funzione.

La tecnica del RL si basa sul presupposto che all'interno di un sistema si possano predisporre [6]:

1. un meccanismo logico A in grado di scegliere degli output sulla base degli input ricevuti.
2. un meccanismo logico B in grado di valutare l'efficacia degli output rispetto ad un preciso parametro di riferimento.
3. un meccanismo logico C capace di cambiare il meccanismo A per massimizzare la valutazione di efficacia effettuata da B.

Il modo in cui questi meccanismi dovrebbero collaborare è descritto dai seguenti punti:

- Se il meccanismo A effettua una scelta efficace allora il meccanismo B manda in output un *premio* proporzionale all'efficacia della scelta di A.
- Se il meccanismo A effettua una scelta inefficace allora il meccanismo B manda in output una *penalità* proporzionale all'inefficacia della scelta di A.
- Il meccanismo C, osservando l'agire di A e B, cerca di modificare la funzione matematica che regola il comportamento di A in modo da massimizzare la quantità e la qualità dei "premi".

I meccanismi B e C sono quelli che vanno a costituire il metodo di *rinforzo* proprio di questa metodica di apprendimento.

Per attuare i meccanismi ed i comportamenti descritti nelle righe precedenti, dal punto di vista logico, si necessita delle seguenti componenti:

- *Insieme di Input*: rappresenta i possibili input che il sistema può ricevere (servono per determinare lo stato del sistema).
- *Funzione valore di stato*: associa un parametro di valutazione ad ogni stato del sistema.
- *Funzione valore di azione*: associa un parametro di valutazione ad ogni possibile coppia stato-azione.
- *Tecnica di rinforzo*: consiste in una *funzione di rinforzo* che, a seconda delle prestazioni attuali e dell'esperienza passata, fornisce delle direttive con cui cambiare la *funzione di valore di stato* e la *funzione di valore d'azione*.
- *Insieme di Output*: rappresenta le possibili decisioni che il sistema può intraprendere.

Gli *input* al sistema possono provenire dai più svariati sensori. Ad esempio, nel caso di un robot che deve imparare a muoversi all'interno di un percorso, gli input potrebbero essere forniti da dei sensori di prossimità che dovrebbero essere poi ri-mappati in opportuni stati che nel caso di questo esempio potrebbero essere "ostacolo di fronte", "strada libera", "muro sul lato" ecc.

[6] Sezione 8, *Apprendimento con rinforzo* di A. Bonarini, M. Matteucci, Politecnico di Milano.

La *funzione valore di stato* V: $S \rightarrow \mathbb{R}$ è quella che ad ogni stato identificato dal sistema e determinato sulla base degli input, associa un valore relativo al grado di bontà della situazione.

La *funzione di valore di azione* è quella che ad ogni coppia composta da stato e azione associa un valore relativo al grado di bontà della combinazione; viene generalmente espressa nella forma:

Q: *S x A* → $\mathbb{R}$

Un agente per il RL può includere anche:

- Una *policy*: è una distribuzione di probabilità per le azioni dati gli stati, cioè la funzione del comportamento dell'agente oppure come l'agente sceglie le azioni dato che si trova in un certo stato.
  Potrebbe essere una policy deterministica, che si vuole imparare dall'esperienza → $a = \pi(s)$
  oppure stocastica: $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$.

- Una *Value function:* è una funzione che esprime il grado di bontà di una determinata azione / stato o, detto in altro modo, quanto sia 'buono' trovarsi in un determinato stato o scegliere una particolare azione. Essa informa l'agente di quale ricompensa finale aspettarsi nello scegliere una particolare azione in un dato stato. In breve, è una previsione delle ricompense future attese, che viene usata per valutare quanto sia 'buono', o meno, uno stato, e consentire all'agente di effettuare la scelta dell'azione, sulla base di ciò, migliore. La value function può esprimersi come:
  $V_\pi(s) = \mathbf{E}\,(R_{t+1} + \gamma\, R_{t+2} + \gamma^2\, R_{t+3} + \ldots \mid S_t = s)$
  Dunque: per un dato stato S, relativo all'istante di tempo t, la value function restituisce (informa l'agente su) i reward cumulati attesi, sotto una certa policy π, così da consentire la scelta dell'azione che massimizza tale somma attesa.
  La value function, com'è chiaro adesso, dipende dal modo in cui l'agente si comporta.
  - $\gamma \in [0, 1]$ è un fattore di sconto: dà indicazione all'agente di quanto dovrebbe importargli dei reward immediati piuttosto che di quelli futuri.
    γ=0 significa che l'agente ha una vista strettamente a breve termine (potremmo definirlo 'miope'), poiché guarda solo al reward immediato.
    Al contrario, γ=1 significa che l'agente ha una vista strettamente a lungo termine, perché tiene conto solo dei reward futuri (tutti allo stesso modo).
    Al di là dei due casi estremi, vi sono tutti i modi intermedi tra i due per impostare il parametro di sconto γ: più è basso, più si vuole dare importanza ai reward prossimi allo stato corrente S_t e più è alto, più si vuole invece dare importanza al futuro, quindi ai reward che ci si aspetta di ottenere nel lungo termine.

- Un *modello*: predice le risposte dell'ambiente. È la rappresentazione da parte dell'agente, dell'ambiente, cioè come l'agente pensa che l'ambiente funzioni. Avremmo una funzione di transizione

$\mathcal{P}$, che predice lo stato successivo o le dinamiche dell'ambiente: $\mathcal{P}_{ss'}^{a} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$.

Si tratta della distribuzione di probabilità sui possibili stati successori, dato lo stato attuale e l'azione intrapresa dall'agente.

Si può dunque cercare di capire le dinamiche sottostanti ed impostare un modello.

Il modello deve includere una funzione ricompensa R, che predice quale sia il reward immediato associato ad una data azione, dato lo stato corrente: $R_s^a = \mathbb{E}(R_{t+1} | S_t = s, A_t = a)$.

Due elementi rendono RL potente: l'utilizzo di campioni per ottimizzare la performance e l'utilizzo dell'approssimazione di funzione per avere a che fare con ambienti di grandi dimensioni. Grazie a queste due component chiave, l'apprendimento con rinforzo può essere utilizzato in ambienti di grandi dimensioni nelle seguenti situazioni:

- un modello dell'ambiente è noto, ma non vi è una soluzione analitica disponibile;
- solo un modello dell'ambiente da simulazione è dato (il tema dell'ottimizzazione basata su simulazione o '*simulation-based optimization*') [7];
- l'unico modo per collezionare informazione sull'ambiente è interagirci.

I primi due problemi potrebbero essere considerati problemi di pianificazione, dato che qualche forma di modello è disponibile, mentre l'ultimo potrebbe essere considerato essere un genuino problema di *learning*. Ad ogni modo, RL converte entrambi i problemi di pianificazione in problemi di apprendimento automatico.

2.2.1 Rappresentazione dello stato

Anche solo da quanto visto sinora, appare evidente la centralità del concetto di stato, e dunque il modo in cui rappresentarli, per cui ci si chiede quale sia il modo più opportuno per rappresentare uno stato del sistema.

Utilizziamo, per rappresentare lo stato dell'agente, quello che chiamiamo *stato Markov*, ad indicare che riassume tutta l'informazione utile della storia antecedente, ed in più possiede la proprietà di Markov.

Uno *stato* è *Markov* se e solo se: $\mathbb{P}(S_{t+1} | S_t) = \mathbb{P}(S_{t+1} | S_1, .., S_t)$.

Questo significa che uno stato dell'agente è Markov se contiene tutte le informazioni rilevanti che l'agente ha ottenuto fino all'istante considerato, il che si traduce nel fatto che si può evitare di tenere in considerazione tutti gli stati precedenti, ma invece guardare solo a quello corrente.

Se vale la proprietà di Markov, il futuro è indipendente dal passato, dato il presente. Dunque, una volta noto lo stato corrente, la storia può essere scartata e lo stato corrente è una statistica sufficiente a darci la stessa caratterizzazione del futuro che ci darebbe considerare l'intera storia passata.

[7] Gosavi, Abhijit (2003). *Simulation.based Optimization: Parametric Optimization Techniques and Reinforcement.* Operations Research/Computer Science Interface Series. Springer. ISBN 978-1-4020-7454-7.

Nel caso in cui l'ambiente è completamente osservabile, per cui l'agente è in grado di osservare direttamente lo stato dell'ambiente come risultato, l'osservazione emessa dall'ambiente è il nuovo stato dell'agente, oltre che il nuovo stato dell'ambiente. In questo caso, -che è quello che consideriamo in questa tesi-, si tratta di un *Markov Decision Process* (MDP), che abbiamo introdotto teoricamente nel capitolo 1.

Nel caso contrario invece, ossia quello in cui l'ambiente sia solo parzialmente osservabile, per cui l'agente effettua un'osservazione dell'ambiente solo indiretta (ad esempio, un robot con una telecamera impostata a cui non viene passata l'informazione sulla sua posizione esatta, ma tutto ciò che conosce è ciò che vede davanti a lui); ecco, in questo caso, lo stato dell'agente è differente dallo stato ambiente, e formalmente, si tratta di un *Partially Observable Markov Decision Process* or (POMDP).

Nei POMDP l'agente deve trovare un modo di costruire la propria rappresentazione dello stato: come detto in precedenza, per 'costruire' lo stato corrente si potrebbe usare la storia nel suo intero, ma è un approccio inefficiente. Si può invece utilizzare la credenza dell'agente circa l'ambiente, ossia ciò che lui pensa che sia in quel momento; dove ha fiducia di essere, internamente all'ambiente. Messo in formule: $S_t^a = (\boldsymbol{P}(S_t^e = s^1), \ldots, \boldsymbol{P}(S_t^e = s^n))$. Questo è un approccio bayesiano, in cui vi è una distribuzione di probabilità per 'dove l'agente pensa di essere all'interno dell'ambiente'.

Oppure, si potrebbe usare una rete neurale ricorrente (RNN), per costruire lo stato dell'agente corrente: $S_t^a = \sigma(S_{t-1}^a W_s + O_t W_0)$. In questo modo, si considera una combinazione lineare dello stato precedente, moltiplicata per un certo peso, e l'osservazione corrente, moltiplicata per un altro dato peso; alla combinazione lineare viene applicata una funzione non lineare espressione di ciò che "crede" l'agente, e si ottiene così lo stato dell'agente corrente.

2.2.2 Ricompensa (reward)

Una ricompensa R_t è un segnale di feedback scalare, che indica quanto bene stia agendo l'agente al tempo t.

L'agente ha il fine di massimizzare la somma attesa di tutti i reward che otterrà; il reinforcement learning si basa infatti sull' *ipotesi del reward*, che afferma che "tutti gli obiettivi possono essere descritti attraverso la massimizzazione dei reward cumulati attesi".

Ad esempio, se si vuole che l'umanoide cammini, si può suddividere l'obiettivo in una ricompensa positiva in caso di movimento in avanti ed una negativa in caso di caduta. L'agente apprenderà che la ricompensa negativa è associata alle azioni che gli causano di cadere, e alla fine imparerà che non conviene scegliere quelle azioni, ma che dovrebbe piuttosto scegliere quella che gli fa ottenere il reward positivo.

Dunque, l'agente deve selezionare le azioni che massimizzano i reward futuri; ma non basta che l'agente scelga in ogni istante temporale quell'azione che gli dà la ricompensa immediata più alta, in quanto alcune azioni però possono avere delle conseguenze sul lungo termine. I reward non sono tutti immediati, anzi, alcuni sono ritardati, per cui l'agente non può essere "greedy" ("avido") in ogni istante temporale, ossia scegliere sempre l'azione che gli dà il massimo reward immediato (cioè all'istante in cui si trova), ma deve fare una pianificazione preventiva su quale sia l'azione migliore da intraprendere in un certo istante, anche

in vista del futuro. Potrebbe essere conveniente infatti, per l'agente, sacrificare una ricompensa immediata per ottenerne una maggiore sul lungo termine, o viceversa.

2.2.3 Tecnica di rinforzo

A seconda di come si progetta e si decide di implementare il sistema di apprendimento, possono essere utilizzate diverse *funzioni di rinforzo* per cambiare la *funzione valore di stato* e diverse politiche per determinare *premi* e *penalità*.

Dal punto di vista modellistico tutte le *funzioni di rinforzo* possono essere ricondotte alla seguente formula base:

$V_{t+1} = (1 - \alpha)V_t + \alpha\Delta_{t+1}$,

dove $0<\alpha\leq 1$ e Δ_{t+1} è il "premio" o la "penalità" che è stata associata alla corrente azione da parte della *funzione di azione*.

Questa funzione, come si vede dalla formula, altera la *funzione di valore di stato* a partire dal prossimo istante in cui verrà richiamata e in base alla valutazione dell'azione corrente effettuata dalla politica di *premio* (o di *penalità*).

Le più diffuse politiche di *premio* (o di *penalità*) sono:

- *Rinforzo con premio ad orizzonte infinito*: il rinforzo ha sempre la stessa intensità ed è valutato per tutti gli istanti temporali. $\mathbb{E}\left[\sum_{k=0}^{\infty} r_{t+k+1}\right]$
- *Rinforzo con premio ad orizzonte finito*: il rinforzo ha sempre la stessa intensità ed è valutato per un periodo di tempo limitato. $\mathbb{E}\left[\sum_{k=0}^{T} r_{t+k+1}\right]$
- *Rinforzo con premio medio*: il rinforzo ha intensità via via decrescente ma viene valutato per tutti gli istanti temporali. In pratica man mano che il tempo passa, i valori di rinforzo vengono attenuati dando più importanza alle valutazioni effettuate negli istanti iniziali. $\mathbb{E}\left[\lim_{n\to\infty}\left(\sum_{k=0}^{n} r_{t+k+1}\right)\right]$
- *Rinforzo con premio scontato*: il rinforzo è distribuito per tutti gli istanti temporali ma aumenta/diminuisce a seconda di un parametro legato agli istanti temporali in cui viene applicato. $\mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}\right]$

2.2.4 Categorizzazione del comportamento dell'agente

In base a ciò su cui si basa l'agente per prendere le decisioni in merito alle azioni da intraprendere, possiamo determinare varie categorie di comportamento:

- *agente 'Value based'* → valuta tutti gli stati nello spazio degli stati S_t, mentre la policy non viene esplicitata; è la value function a dire all'agente quanto è buona ciascuna azione in un determinato stato ed esso sceglie la migliore.
- *agente 'Policy Based'* → la policy non è più implicita, ma viene esplicitata; l'agente cerca l'ottimo di azione-value function, la cui scelta gli consente di agire in modo ottimo.
- *agente 'Actor-Critic'* → è un agente sia value-based che policy-based: guarda alla policy, ma anche a quanta ricompensa può ottenere da ciascuno stato.
- *agente 'Model-Based'* → l'agente basa il suo comportamento sul modello (che preventivamente cerca di costruire, come spiegato quando abbiamo introdotto il 'modello', ossia l'agente cerca di costruire un modello che descriva il comportamento dell'ambiente, e poi pianifica il comportamento migliore da seguire).
- agente 'Model-Free' → al contrario del precedente, in questo caso l'agente non cerca di capire l'ambiente, di costruirne le dinamiche, ma piuttosto valuta direttamente la policy e/o la value function. Dall'esperienza, si cerca di immaginare una policy che consenta di comportarsi in modo ottimale in vista dell'ottenimento del massimo valore totale dei reward.

2.2.5 Exploration vs Exploitation

Il *trade-off* tra 'esplorazione' e 'sfruttamento', -a cui abbiamo accennato in precedenza-, è stato studiato a fondo attraverso il problema del bandito multi-armato e per MDP a stati finiti in Burnetas and Katehakis (1997). [8]

L'apprendimento per rinforzo richiede intelligenti meccanismi di esplorazione; selezionare le azioni in maniera casuale, senza riferimento ad una distribuzione stimata di probabilità, dimostra una performance povera.

Il caso di (piccoli) MDP finiti è relativamente ben compreso. Ad ogni modo, a causa della mancanza di algoritmi capaci di scalare bene con il numero degli stati, -o di scalare a problemi con spazi degli stati infiniti-, i metodi di semplice esplorazione sono i più pratici.

Un tal metodo è ε-greedy, dove $0<\varepsilon<1$ è un parametro che controlla la quantità di esplorazione vs. sfruttamento: con probabilità 1-ε, è scelto lo sfruttamento, e l'agente sceglie l'azione che crede abbia il miglior effetto a lungo termine (i legami tra le azioni sono interrotti in modo casuale uniforme).

Alternativamente, con probabilità ε, viene scelta l'esplorazione, e l'azione è scelta in modo casuale uniforme.

ε è di solito parametro fissato, ma può venire aggiustato o in base ad uno schedule, -facendo esplorare l'agente progressivamente meno-, o adattivamente in base ad euristiche. [9]

[8] Burnetas, Apostolos N; Katehakis, Michael N. (1997), "Optimal adaptive policies for Markov Decision Processes", *Mathematics of Operations Research,* **22**: 222-255, doi:10.1287/moor.22.1.222.

2.2.6 Output

L'output consiste in una delle azioni che il sistema può intraprendere, e di conseguenza, in senso esteso, nel comportamento dell'agente.

2.2.7 Supervisione

Come anticipato nell'introduzione del capitolo, nel RL non vi è supervisione, ma solo una ricompensa (un valore reale) che dice all'agente quanto buona (o meno) sia stata l'azione che ha intrapreso.

I feedback da parte dell'ambiente potrebbero essere ritardati di qualche istante temporale: non sono necessariamente istantanei. Ad esempio, per l'obiettivo di raggiungere un certo punto in una griglia, il feedback spesso viene impostato sull'obiettivo, ossia l'agente ottiene la ricompensa solamente alla fine del suo percorso (se risulta nel raggiungimento del goal). L'agente probabilmente spenderà del tempo esplorando l'ambiente; vagando, finché non riesce a raggiungere l'obiettivo (per caso, o quando ha realizzato quali azioni sono state buone e quali no).

Negli altri tipi di apprendimento automatico, l'ambiente viene fornito mediante un dataset all'algoritmo, che cercherà di generalizzare e adattare quello che conosce anche a nuove situazioni, che gli venissero proposte. Nel RL invece, i dati sull'ambiente non sono i.i.d. (indipendenti e identicamente distribuiti): l'agente potrebbe spendere anche molto tempo in certe parti dell'ambiente e non vedere per niente delle altre, le quali potrebbero anche essere interessanti e importanti, al fine che esso impari il comportamento ottimale. Dunque, nel RL il tempo è molto importante: l'agente deve esplorare quasi completamente l'ambiente, per essere in grado di prendere le decisioni migliori.

L'agente, attraverso le azioni che compie, potrebbe anche influenzare l'ambiente, che di conseguenza manderà un certo segnale: si tratta di un processo di apprendimento attivo e dinamico.

2.3 Formalizzazione del problema di RL

Quasi tutti i problemi di Reinforcement Learning possono essere formalizzati tramite processo decisionale di Markov (MDP) (per quei problemi per cui l'ambiente risulta totalmente osservabile, come abbiamo già spiegato): il processo di cui abbiamo parlato, che vede l'agente osservare l'output mandato dall'ambiente, consistente in una ricompensa ed in uno stato nuovo, ed agire in base a ciò ricade nel modello matematico del MDP.

I MDP, infatti, sono un'inquadratura perfetta per il problema dell'apprendimento tramite interazione, per il conseguimento di un obiettivo. L'agente e l'ambiente interagiscono continuamente, il primo seleziona azioni e il secondo risponde ad esse presentando nuove situazioni ed emettendo delle risposte di bontà della scelta.

---

[9] Tokic, Michel; Palm, Günther (2011), "Value-Difference Based Exploration: Adaptive Control Between Epsilon-Greedy and Softmax" (PDF), *KI 2011: Advances in Artificial Intelligence,* Lecture Notes in Computer Science, **7006**, Springer, pp. 335-346, ISBN 978-3-642-24455-1.

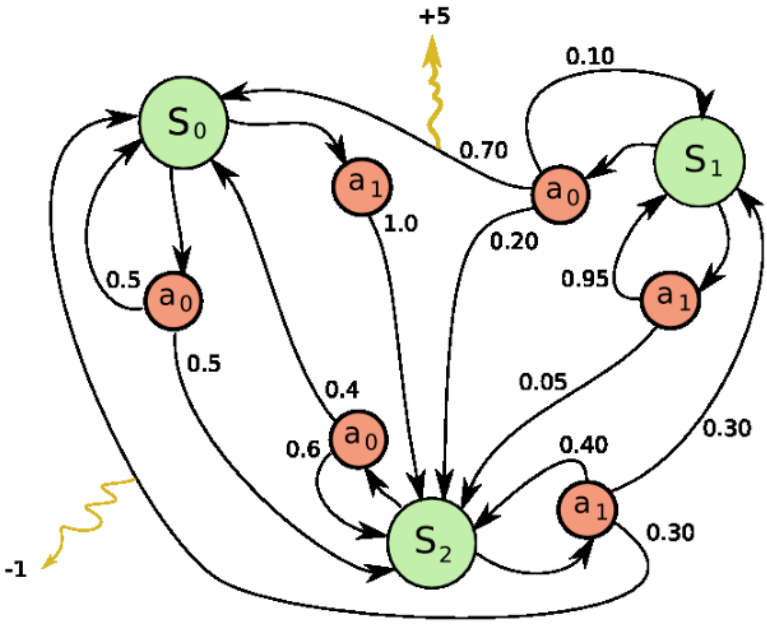


FIGURA 2. 3 MARKOV DECISION PROCESS

In figura 2.3 abbiamo un esempio molto semplice di come funziona un MDP: vediamo che sono presenti stati, azioni e ricompense (e probabilità). Dallo stato inziale S_0 si può scegliere l'azione a_1, che porta inequivocabilmente allo stato S_2, poiché vediamo che ha probabilità certa (1.0) di portare ad S_2, o l'azione a_0, che con pari probabilità può far rimanere nello stato iniziale o far giungere ad S_2. Giunti in S_2 si può scegliere l'azione a_1, che ha probabilità 0.30 di far ritornare indietro ad S_0, -ed in questo caso si riceverebbe un reward negativo pari a '-1'-, 0.40 di far rimanere in S_2 (per cui all'istante successivo si dovrà effettuare la stessa identica scelta), e altri 0.30 di andare in S_1, e così via.

Notiamo che l'unico reward positivo si riceve nel passare dallo stato S_1 allo stato S_0 mediante la scelta dell'azione a_0. L'agente per il RL dunque, in questo caso deve capire che il suo obiettivo è ritornare allo stato iniziale dopo aver visitato gli altri due, cercando di evitare di prendere il reward -1 selezionando l'azione a_0 quando si trova nello stato S_2 (che è impossibile a livello probabilistico da evitare completamente): deve capire che deve fare ciò nel modo più efficace possibile.

Riprendiamo gli MDP, visti a livello teorico nel capitolo 1:

– è innanzitutto fondamentale la *proprietà di Markov,* che afferma sostanzialmente che "il futuro è indipendente dal passato, dato il presente". → Noto lo stato corrente, la storia informative incontrata fino a quel momento può essere scartata, in quanto lo stato attuale è una statistica sufficiente, che ci dà la stessa caratterizzazione del futuro che ci dà la storia nel suo intero.

In termini matematici, uno stato S_t ha la proprietà di Markov se e solo se: $\mathbb{P}(S_{t+1}|S_t) = \mathbb{P}(S_{t+1}|S_{1,\dots,}S_t)$ → lo stato cattura tutta l'informazione rilevante dalla storia passata.

Da uno stato Markov S ad un suo stato successore S', la funzione di probabilità di transizione di stato è definita da: $\mathcal{P}_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s)$. → È una distribuzione di probabilità sopra i possibili stati successori, dato lo stato corrente, ossia l'agente si trova in un certo stato, vi è una certa probabilità di andare (all'istante successivo) al primo stato, un'altra di andare al secondo e così via…

Questa funzione di transizione può essere espressa in forma di matrice, in cui, per la teoria della probabilità, ogni riga somma a 1. Supponendo di avere n stati possibili nel problema: $\mathcal{P} = \begin{pmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{pmatrix}$.

– *Processo markoviano* → è un processo casuale senza memoria, vale a dire una sequenza di stati casuali $S_1, S_2, \ldots$ che godono della proprietà di Markov.

Un processo markoviano, o catena di Markov (visto che supponiamo spazio degli stati discreto), è una tupla $(S, \mathcal{P})$ sullo spazio degli stati S, con funzione di transizione $\mathcal{P}$: le dinamiche del sistema possono essere definite dalle due componenti S e $\mathcal{P}$.

- Campionando da un MDP, si estrae una sequenza di stati come appena specificato, che chiamiamo *episodio*.

- Vengono indicati lo stato iniziale e lo/gli stati terminali (cioè quelli in cui termina il processo).

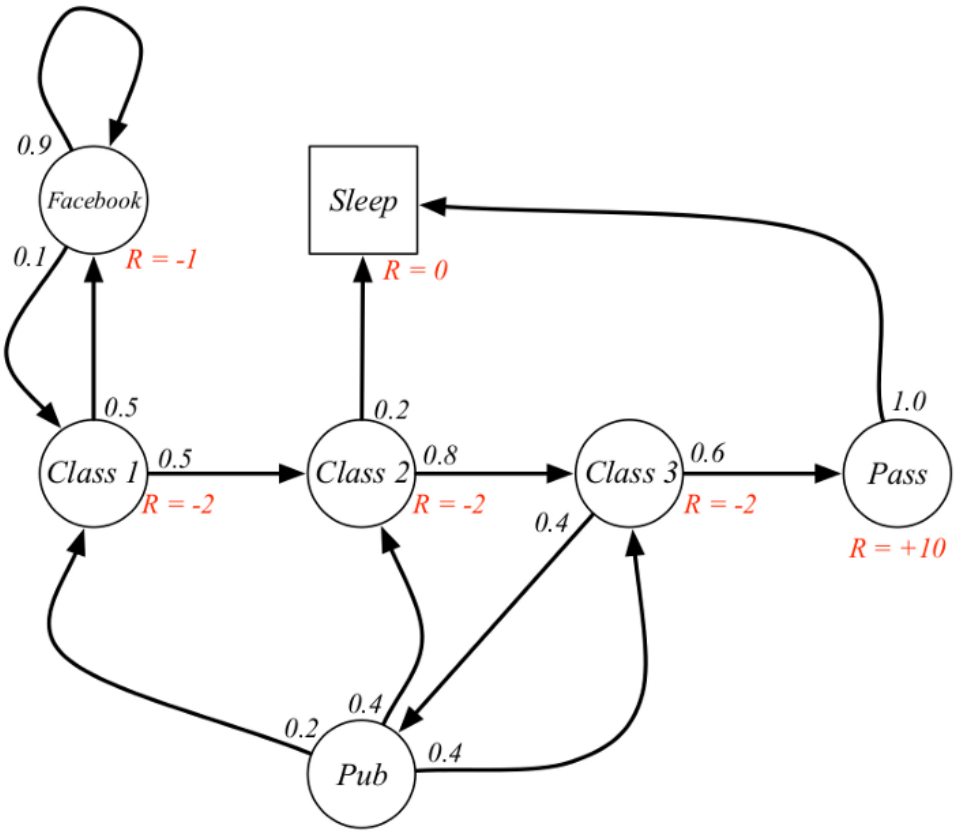


FIGURA 2. 4 ESEMPIO DI MRP DALLA LEZIONE DI DAVID SILVER N.1

Nell'esempio di figura 2.4 abbiamo diversi stati con diversi successori; ad esempio, in 'Class 1' si può andare alla 'Class 2' oppure su 'Facebook' con probabilità 0.5.
Un episodio sarebbe ad esempio: "Class 1 →Class 2 → Class 3 → Pass →Sleep".
'Sleep' è il nodo terminale, che fa sì che si concluda un episodio, a prescindere da come ci si arrivi.

- Una catena di Markov a cui si aggiunge un giudizio sui valori è un *MRP* (Markov Reward Process), dove il giudizio 'attesta' quanta ricompensa è stata accumulata nella particolare sequenza che è stata campionata.

Un MRP è una tupla $(S, \mathcal{P}, \mathrm{R}, \gamma)$, dove S è uno spazio (che consideriamo sempre finito) delle azioni, $\mathcal{P}$ è la funzione di probabilità di transizione di stato, R è una funzione di ricompensa per cui: $R_s = \mathbb{E}(R_{t+1}|S_t = s)$, la quale ci dice quanta ricompensa immediata aspettarci di ottenere dallo stato S nel momento corrente.

- Inseriamo la nozione di *ritorno* G_t, che rappresenta la somma (eventualmente scontata) delle ricompense ottenute a partire dall'istante t; G_t è ciò che interessa massimizzare, ed è:

$G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, dove γ è un fattore di sconto, $\gamma \in [0,1]$, che informa l'agente di quanto dovrebbe interessarsi dei reward immediati verso quelli futuri, come spiegato nel paragrafo 2.2.

Ad ogni modo, ciò che è importante sottolineare in questo punto, -senza entrare nel dettaglio di come scegliere γ e di cosa un γ alto piuttosto che basso significa (posizione già introdotta nello stesso paragrafo 2.2)-, è che l'agente è interessato alla somma dei reward che otterrà, non ad uno o a qualche reward.

La funzione del fattore di sconto è: innanzitutto quella di essere conveniente da introdurre, matematicamente, perché garantisce la convergenza dell'algoritmo (per cui si evitano a priori reward infiniti in catene di Markov con *loops*). E facciamo notare che, anche trattandosi di una somma infinita, per $\gamma < 1$ e reward costante, risulta finita; ad esempio, per reward pari a 1, si riconduce alla celebre serie geometrica: $G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$.

Un'altra ragione per scontare i reward è che l'agente non è certo di ciò che accadrà nel futuro: potrebbe essere meglio prendere il reward immediato piuttosto che aspettare nella speranza di prendere uno più grande nel futuro, quindi γ definisce in un certo senso 'fino a che orizzonte guardare' per ciò di cui interessarsi.

È possibile usare MRP in cui non vi è un fattore di sconto (ossia $\gamma = 1$), se si ha la certezza che tutte le sequenze termineranno.

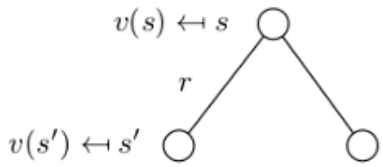
Abbiamo precedente introdotto anche il concetto di *value function*, che ha la funzione di dire quanto 'buono' o 'cattivo' sia un certo stato e/o una certa azione, ossia quanto sia positivo essere in un certo stato o prendere una certa decisione di azione. Informa l'agente di quanta ricompensa aspettarsi scegliere una certa azione in un dato stato. La funzione valore di stato di un MRP è il ritorno atteso a partire dallo stato s: $v(s) = \mathbb{E}(G_t|S_t = s)$ (*); ad esempio, il ritorno dell'episodio di fig. 2.4 che abbiamo considerato poco fa

(Class 1 →Class 2 → Class 3 → Pass →Sleep) sarebbe, con un $\gamma = 0.5$: $v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$=-2.25. Per cui il valore dello stato 'classe1' sarebbe pari a -2.25.

- L'agente cerca di ottenere la più alta somma di reward attesa da ogni stato in cui passa; ciò significa che cerca la value function ottima, ossia la somma di reward cumulati massima. In tal senso è utile *l'equazione di Bellman* → la value function viene decomposta in due parti: un reward immediato R_{t+1}, e il valore scontato dello stato successivo $\gamma V(S_{t+1})$, con $v(s)$ dato da (*). Sviluppando il ritorno totale atteso: $G_t = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s]$ e sostituendo il ritorno G_{t+1}(che è quello a partire dall'istante $t+1$, otteniamo: $G_t = \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s]$.
Infine, dato che il valore atteso è una funzione lineare, cioè $\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$, il valore atteso del ritorno G_{t+1} è il valore dello stato S_{t+1}, per cui: $G_t = \mathbb{E}[R_{t+1} + \gamma\, v(S_{t+1}) | S_t = s]$.

Otteniamo dunque l'*equazione di Bellman per gli MRP*: $v(s) = \mathbb{E}[R_{t+1} + \gamma\, v(S_{t+1}) | S_t = s]$. (1)

Quindi, per ogni stato nello spazio degli stati, l'equazione di Bellman ci dà il valore di quello stato:



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

FIGURA 2.5 VALORE DELLO STATO MEDIANTE EQ. DI BELLMAN PER MRP

L'equazione significa che il valore dello stato S è dato dal reward che otteniamo quando lasciamo quello stato, più una somma scontata sui successivi possibili stati, dove il valore di ciascun possibile successore è moltiplicato per la probabilità di giungervi.

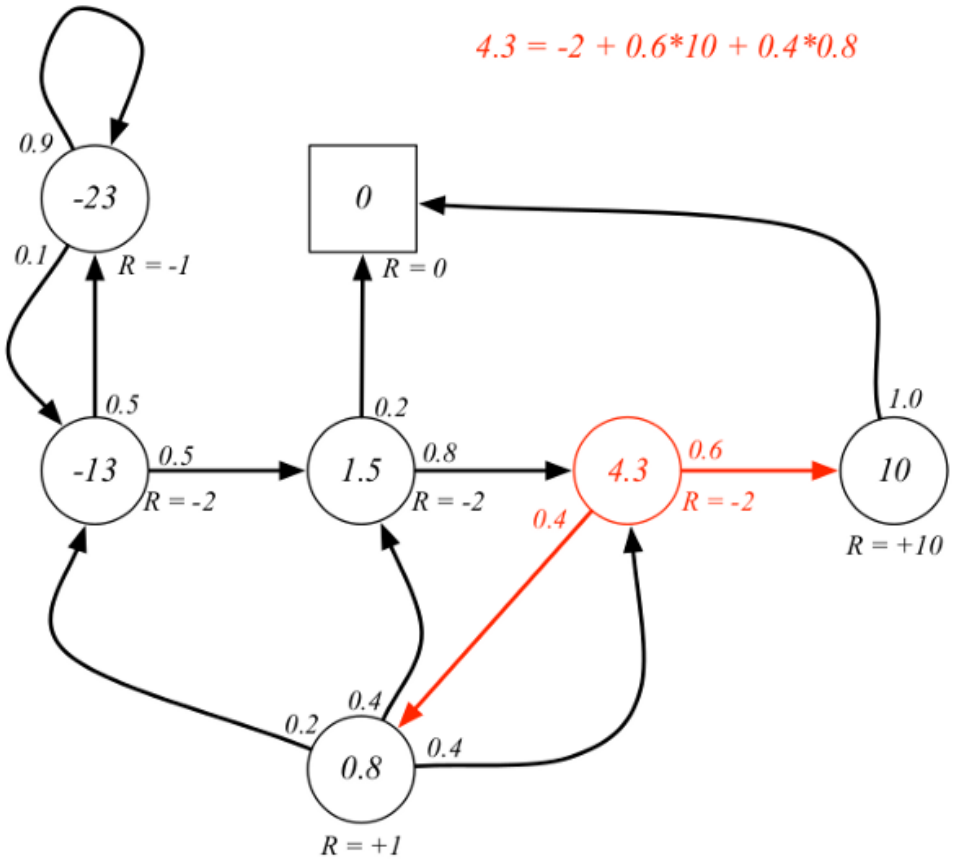


FIGURA 2.6 ESEMPIO DI MRP DALLA LEZIONE DI DAVID SILVER N.2

In figura 2.6 sono indicati i valori degli stati per l'esempio che abbiamo considerato prima; ad esempio, abbiamo evidenziato il valore della classe 3 per esplicitare il calcolo: al reward immediato che si prende nello stato 'Class3' (-2) si somma '10' che si ottiene se si giunge nello stato 'Pass', moltiplicato per 0.6 che è la probabilità di giungerci, e '0.8', che si ottiene se giunge nello stato 'Pub', anch'esso moltiplicato per la probabilità di giungervi, che per esso è 0.4. Precisiamo che abbiamo supposto un γ pari ad uno, ma le somme sui possibili stati successori, come sappiamo dalla formula, sono in linea di principio scontate.

Questa equazione di Bellman è lineare, per cui può essere risolta direttamente:

$$v = \mathcal{R} + \gamma \mathcal{P} v$$
$$(I - \gamma \mathcal{P})\, v = \mathcal{R}$$
$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

La complessità computazionale di questa soluzione è $O(n^3)$ per n stati, motivo per cui questa soluzione diretta è possibile solo per MRP picccoli. Per problemi più grandi, si utilizzano metodi iterativi, come la Programmazione Dinamica, la valutazione Monte-Carlo, e il Temporal Difference learning (ovvero l'apprendimento mediante differenza temporale).

– Un *MDP finito* (come nel nostro interesse) è un Markov Reward Process con l'aggiunta delle decisioni; tutti gli stati sono Markov, ed è il modo in cui rappresentiamo il problema di RL che vorremo andare a risolvere.

Un MDP finito è una tupla $(S, A, \mathcal{P}, R, \gamma)$, dove S è uno spazio finito di stati, A è uno spazio finito di azioni, $\mathcal{P}$ è la funzione di transizione di probabilità $\mathcal{P}^{a}_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$, R è una funzione di ricompensa tale per cui $R^a_s = \mathbb{E}(R_{t+1} | S_t = s, A_t = a)$, e γ è un fattore di sconto, $\gamma \in [0,1]$.

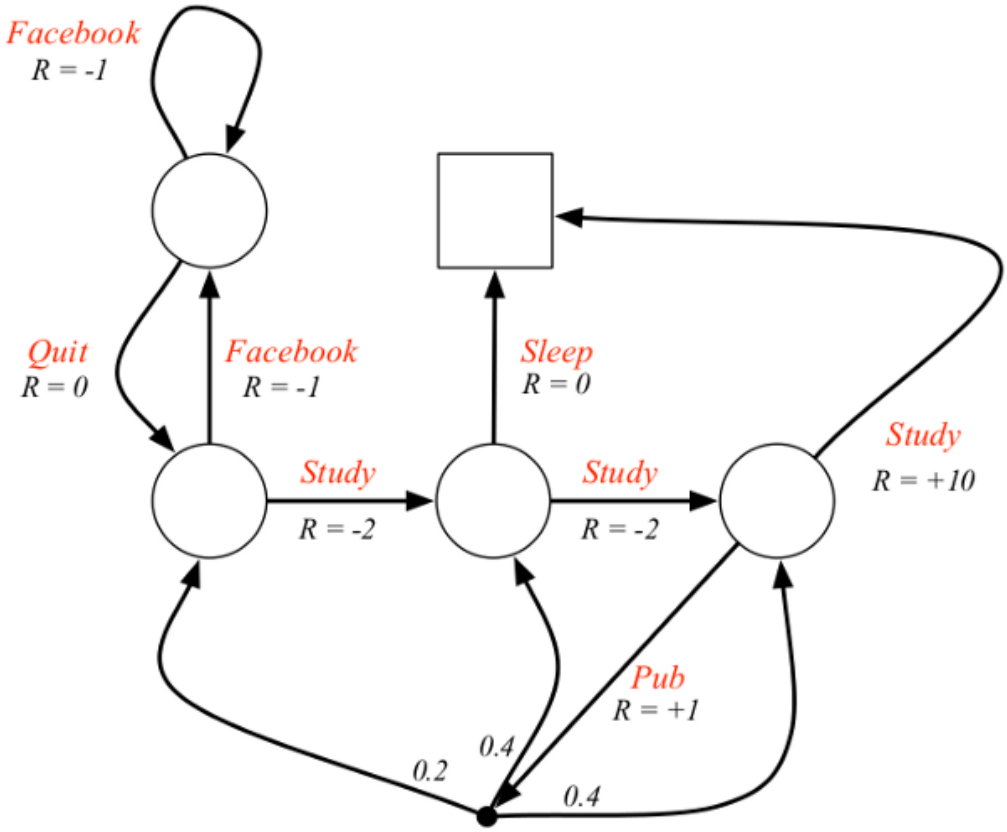


FIGURA 2.7 ESEMPIO DI MDP DALLA LEZIONE DI DAVID SILVER

Ricordiamo che in questo caso una policy π è una distribuzione sulle azioni, dati gli stati; definisce il comportamento di un agente, e in formule è definita in questo modo: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$ $\pi: A\ x\ S \rightarrow [0,1]$.

Le policy per gli MDP dipendono dallo stato corrente, non dalla storia, cioè sono *stazionarie*: $At \sim \pi(\cdot | St), \forall t > 0$, il che significa che quandunque l'agente arrivi in un particolare stato, sceglierà l'azione che la policy ha precedentemente deciso, per tutti gli step temporali.

◦ Le policy possono essere stocastiche per consentire di fare esplorazione nello spazio degli stati.

- Possiamo sempre ricavare un processo Markoviano o un MRP da un MDP: dato un MDP M=$(S, A, \mathcal{P}, R, \gamma)$, e una policy π, la sequenza di stati $S_1, S_2, \ldots$ è un processo Markoviano $(S, \mathcal{P})$, sulla policy π.

La sequenza di stati e ricompense $S_1, R_1, S_2, R_2, \ldots$ è un Markov Reward Process $(S, \mathcal{P}, R, \gamma)$, dove $\mathcal{P}^{\pi}_{s,s'} = \sum_{a \in A} \pi(a|s) \mathcal{P}^{a}_{ss'}$ → si va a fare una media di tutte le cose che potrebbero succedere sotto la nostra policy π. Abbiamo così definito la funzione delle dinamiche di transizione sulla policy π essere la media delle

dinamiche di transizione di tutto ciò che potremmo fare: vi è una certa probabilità di intraprendere l'azione a sotto la policy π, dallo stato S; la moltiplichiamo per 'ciò che succederebbe dopo che l'abbiamo intrapresa', vale a dire lo stato successore in cui potremmo arrivare.

Per quanto riguarda la funzione di ricompensa, il discorso non cambia; la reward function è: $R_s^\pi = \sum_{a\in A} \pi(a|s) R_s^a$. → Si fa la media su tutti i possibili reward associati a diverse possibili azioni, dallo stato S.

- Abbiamo già l'espressione della value function per gli MRP; abbiamo quella della policy per gli MDP, ossia un modo di scegliere, -un comportamento da seguire-, in un processo markoviano.

- La *funzione di stato-valore* $V_\pi(s)$ di un MDP è il ritorno atteso partendo dallo stato S e poi seguendo la policy π.

Ci dice quanto è buono trovarsi nello stato S se si sta seguendo la policy π, cioè le aspettative quando campioniamo tutte le azioni secondo la policy π. È data da:

$v_\pi(s) \coloneqq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s]$, per tutti gli $s \in S$.

- La *funzione di azione-valore* $q_\pi(s,a)$ è il ritorno atteso, partendo dallo stato S, intraprendendo l'azione a, e poi seguendo la policy π.

Ci dice quanto è conveniente intraprendere una certa azione, in un certo stato, ed è data da:

$q_\pi(s,a) \coloneqq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,|S_t = s,\ A_t = a]$

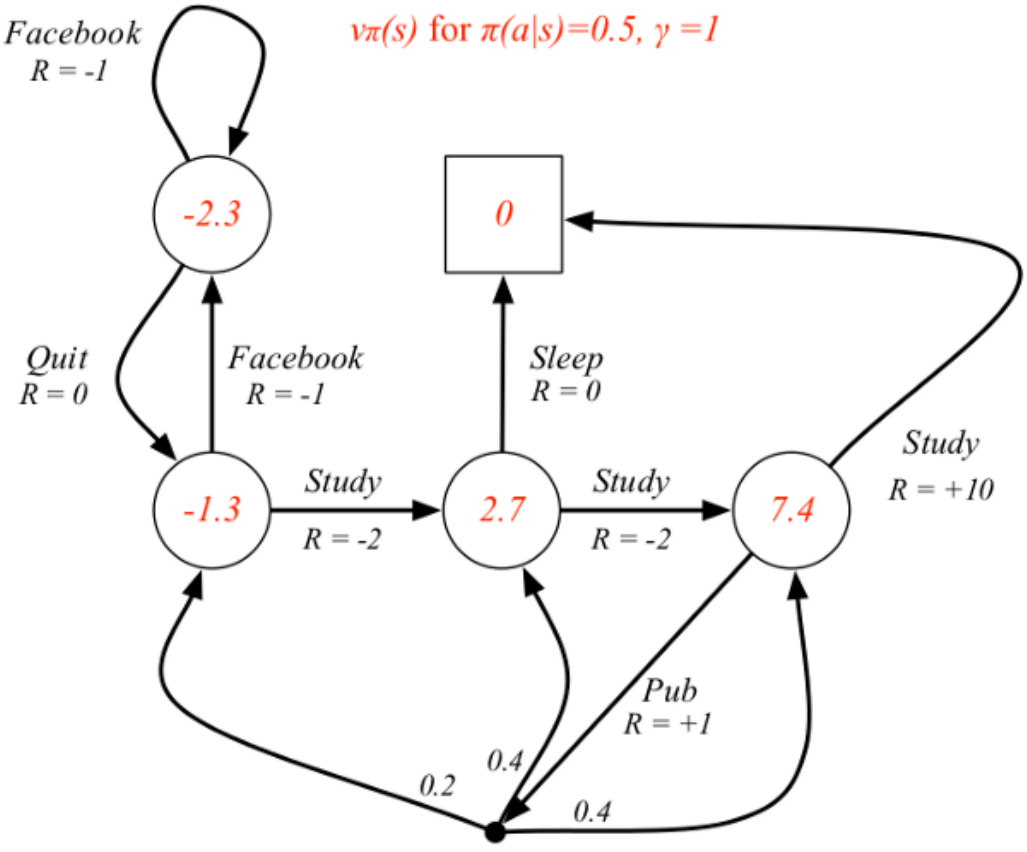


FIGURA 2.8 FUNZIONE DI STATO-VALORE NELL'ESEMPIO DI MDP

Con la figura 2.8 concludiamo l'esempio di David Silver sulle classi, evidenziando la funzione di stato-valore per il particolare MDP e le ipotesi indicate.

2.4 Algoritmi per l'apprendimento del controllo

Nel generico problema di learning, anche assumendo di trascurare il problema dell'esplorazione, e anche se gli stati sono osservabili (cosa che supponiamo), rimane il problema di capire come utilizzare l'esperienza passata per capire quali azioni portano ai reward cumulati maggiori.

Abbiamo già in parte introdotto alcuni dei concetti e metodi che andiamo ora a spiegare più schematicamente e più nel dettaglio, circa come affrontare il problema di learning in sé e per sé.

Nel sottoparagrafo immediatamente successivo richiamiamo per chiarezza e comodità alcune parti fondamentali di quanto detto sinora.

2.4.1 Criterio di ottimalità

La selezione dell'azione da parte dell'agente è modellata come una mappa, detta *policy:*

$\pi: AxS \to [0,1], \qquad \pi(a,s) = \mathbb{P}(a_t = a, s_t = s)$.

Dà la probabilità di intraprendere l'azione a quando si è nello stato s.

Esistono anche policy non probabilistiche.

La *funzione-valore* $V_\pi(s)$ è definita come il ritorno atteso a partire dallo stato s, cioè $s_0 = s$, e successivamente seguendo la policy π. Dunque, approssimativamente, la funzione-valore stima "quanto sia buono" trovarsi in un determinato stato.

$V_\pi(s)$ =$\mathbb{E}$ (R) = $\mathbb{E}$ $[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$, dove la variabile R denota il *ritorno*, ed è definita come la somma delle ricompense future scontate.

R=$\sum_{t=0}^{\infty} \gamma^t r_t$, dove r_t è la ricompensa al passo t; $\gamma \in [0,1)$ è il tasso di sconto: ipotizziamo γ minore di uno, in modo tale che man mano che uno stato diventa vecchio, il suo effetto sugli stati successivi diventa sempre minore, cioè scontiamo il suo effetto.

L'algoritmo deve trovare una politica (*policy*) che dia il ritorno atteso massimo. Dalla teoria degli MDP è noto che, senza perdita di generalità, la ricerca può essere ristretta all'insieme delle cosiddette politiche stazionarie.

Una *politica* è *stazionaria* se la distribuzione delle azioni che restituisce dipende solo dall'ultimo stato visitato (relativamente alla storia delle osservazioni dell'agente).

La ricerca può essere ulteriormente ristretta alle *politiche stazionarie deterministiche*: una tale politica seleziona le azioni basandosi sullo stato corrente deterministicamente, -ossia ripete sempre le stesse scelte, dato il punto in cui si trova; non vi è stocasticità-. Poiché qualsiasi di queste politiche può essere identificata tramite una mappatura dall'insieme degli stati a quello delle azioni, queste politiche possono essere identificate tramite tali mappature senza che vi sia perdita di generalità.

2.4.2 Forza bruta

L'approccio *forza bruta* (o *ricerca esaustiva*) comporta due passi:

- per ogni politica possibile, campionare i ritorni mentre viene seguita;
- scegliere la politica con il ritorno atteso maggiore.

Un problema con questo metodo è che il numero di politiche può essere grande, o anche infinito; un altro è che può essere grande la varianza dei ritorni, il che richiede molti campioni per stimare con precisione il ritorno di ciascuna politica.

Questi problemi possono essere migliorati se si assume una qualche struttura e si consente che i campioni generati da una politica possano influenza le stime fatte per gli altri. I due approcci principali per raggiungere ciò sono la stima della funzione-valore (*value function estimation*) e la ricerca diretta di policy (*direct policy search*).

2.4.3 Funzione Valore

Gli approcci di tipo *Value function* cercano di trovare una politica che massimizzi il ritorno mantenendo un insieme di stime dei ritorni attesi per una qualche politica, solitamente o quella 'corrente', detta '*on-policy*', o quella ottima, detta '*off.-policy*'.

Questi metodi si basano sulla teoria dei MDP, in cui l'ottimalità è definita in un senso che è più forte di quello di poco sopra: una politica è detta ottima se raggiunge il miglior ritorno atteso a partire da qualunque stato iniziale, vale a dire le distribuzioni iniziali non giocano alcun ruolo in questa definizione. Di nuovo, una politica ottimale può sempre essere trovata entro quelle stazionarie.

Per definire l'ottimalità in una maniera formale, definiamo il valore di una politica π come:

$V^{\pi}(s) = \mathbb{E}[R|s,\pi]$, dove R sta per il ritorno associato al seguire π dallo stato iniziale s.

Definiamo $V^{*}(s)$ come il massimo valore possibile di $V^{\pi}(s)$, dove π è consentito che possa cambiare: $V^{*}(s) = max_{\pi} V^{\pi}(s)$.

Una politica che raggiunge questi valori ottimi in ciascuno stato è chiamata *ottima/ottimale*. Chiaramente, una politica che è ottimale in questo senso forte è anche ottimale nel senso che massimizza il ritorno atteso ρ^{π}, dato che $\rho^{\pi} = \mathbb{E}[V^{\pi}(S)]$, dove S è uno stato campionato in modo casuale da una certa distribuzione μ.

Anche se gli stati-valori (valori di stato) sono sufficienti per definire l'ottimalità, è utile definire i valori di azione. Dato uno stato s, un'azione a ed un policy π, il valore di azione della coppia (s,a) sotto π è definito da $Q^{\pi}(s,a) = \mathbb{E}[R|s,a,\pi]$, dove R ora sta per il ritorno casuale associato con l'intraprendere prima l'azione a in s e in seguito seguire π.

La teoria dei MDP afferma che se π^{*} è una politica ottimale, agiamo in modo ottimo, cioè scegliamo l'azione ottima, se scegliamo l'azione da $Q^{\pi^{*}}(s,\cdot)$ con il più alto valore ad ogni stato, s.

La *funzione azione-valore* di una tale politica ottimale (Q^{π^*}) è chiamata la *funzione azione-valore ottimale*, ed è comunemente denotata con Q^*.

In sintesi, conoscere la funzione di azione-valore ottimale è da sé sufficiente per sapere come agire ottimalmente.

Assumendo conoscenza completa del MDP, i due approcci di base per calcolare la funzione di azione-valore ottimale sono l'*iterazione del valore* (*Value Iteration*) e l'*iterazione della politica* (*policy iteration*).

Entrambi gli algoritmi calcolano una sequenza di funzioni $Q_k(k = 1,2,\dots)$, che converge a Q^*.

Calcolare queste funzioni implica calcolare le aspettative sull'intero spazio degli stati, il che è impratico per tutti gli MDP eccetto quelli piccoli (finiti).

→ Nei metodi per il RL le aspettative sono approssimate facendo la media su campioni e usando tecniche di approssimazione di funzione per far fronte alla necessità di rappresentare le funzioni di valore su spazi di stato-azione grandi.

2.4.4 Metodi Monte Carlo

I metodi Monte Carlo possono essere usati in algoritmi che mimano la *policy iteration*. La *policy iteration* consiste di due step: la valutazione della policy e il suo miglioramento.

Monte Carlo è usato nello step di valutazione della policy nel quale, data una policy π stazionaria e deterministica, l'obiettivo è calcolare i valori di funzione $Q^{\pi}(s,a)$ (o una buona loro approssimazione) per tutte le coppie stato-azione (s,a).

Assumendo (per semplicità) che il MDP sia finito, che vi sia sufficiente memoria disponibile per poter contenere i valori di azione e che il problema sia episodico e che dopo ogni episodio ne inizi uno nuovo da un qualche stato iniziale random. Allora, la stima del valore di una data coppia stato-azione (s,a) può essere calcolata mediando i ritorni che sono stati campionati, con origine da (s,a) nel tempo. Dato sufficiente tempo, questa procedura può perciò costruire una precisa stima Q della funzione valore di azione Q^{π}.

Nello step del miglioramento della policy, la policy successiva è ottenuta elaborando una politica 'avida' (*greedy*) rispetto a Q: dato uno stato s questa nuova politica restituisce un'azione che massimizza $Q(s,\cdot)$.

Nella pratica la *lazy evaluation* può differire il calcolo delle azioni massimizzanti a quando saranno necessarie.

Problemi con questa procedura includono:

- La procedura potrebbe spendere troppo tempo valutando una politica sub-ottimale;
- Utilizza i campioni inefficientemente, dal momento che una traiettoria lunga migliora la stima solo della singola coppia stato-azione da cui ha inizio la traiettoria stessa;
- Quando i ritorni lungo la traiettoria hanno varianza alta la convergenza è lenta;
- Funziona solo in problemi episodici;

- Funziona solo per MDP piccoli e finiti.

2.4.5 Metodi differenza temporale

Il problema n.1 di quelli appena detti è corretto consentendo alla procedura di cambiare la politica, - in qualche o in tutti gli stati-, prima che i valori si stabilizzino. Anche ciò può essere problematico dal momento che potrebbe impedire la convergenza.

La maggior parte degli attuali algoritmi agiscono in questo modo, e rappresentano la classe degli *algoritmi di iterazione della politica generalizzata* (*generalized policy iteration*); molti metodi *actor critic* appartengono a questa categoria.

Il secondo problema può essere corretto consentendo alle traiettorie di contribuire a qualsiasi coppia stato-azione che ne fa parte. Ciò potrebbe anche aiutare in parte col problema n.3, anche se una soluzione migliore quando i ritorni hanno alta varianza sono i metodi a differenza temporale (*temporal difference*, *TD*) di Sutton, che si basano sulla equazione ricorsiva, equazione di Bellman. [10] [11]

Il calcolo nei metodi TD può essere incrementale (quando dopo ciascuna transizione la memoria è cambiata e la transizione viene 'buttata via'), o a gruppo (*batch*, quando le transizioni sono raggruppate e le stime sono calcolate una volta basate sul gruppo). I metodi a gruppo, come la differenza temporale ai minimi quadrati,[12] potrebbe riuscire a utilizzare meglio l'informazione nei campioni, mentre i metodi incrementali sono l'unica scelta quando gli altri sono infattibili a causa della loro alta complessità computazionale o di memoria.

Alcuni metodi cercano di combinare i due approcci.

I metodi basati sulle differenze temporali superano anche il problema n.4.

Per affrontare il problema n.5, vengono usati metodi di approssimazione di funzione (*function approximation methods*).

L'approssimazione di funzione di tipo *lineare* comincia con una mappatura $\emptyset$ che assegna un vettore finito-dimensionale ad ogni coppia stato-azione; poi, i valori di azione di una coppia stato- azione (s,a) sono ottenuti combinando linearmente le componenti di $\emptyset(s,a)$ con dei pesi θ:

$\mathrm{Q(s,a)}=\sum_{i=1}^{d} \theta_i \emptyset_i(s,a)$.

L'algoritmo poi aggiusta i pesi, invece di aggiustare i valori associati alle singole coppie stato-azione.

Sono stati esplorati anche metodi basati su idee da statistiche non parametriche, che possono essere viste costruirsi le loro proprie caratteristiche.

[10] Sutton, Richard S. (1984). *Temporal Credit Assignment in Reinforcement Learning* (PhD thesis). University of Massachusetts, Amherst, MA.

[11] Sutton & Barto, 1998. §6, Temporal-Difference Learning.

[12] Bradtke, Steven J.; Barto, Andrew G. (1996). "Learning to predict by the method of temporal differences". *Machine Learning.* **22**: 33-57.

Anche l'iterazione del valore può essere usata come punto di inizio, dando vita all'algoritmo Q-learning e alle sue numerose varianti. [13]

Il problema con l'utilizzare I valori di azione è che potrebbero aver bisogno di stime estremamente precise dei valori di azione concorrenti, che possono essere difficili da ottenere quando i ritorni sono affetti da rumore, anche se questo problema è mitigato in qualche misura dai metodi sulle differenze temporali; usare il cosiddetto 'metodo di approssimazione di funzione compatibile' compromette generalità ed efficienza.

Un altro problema specifico delle TD è dato dal loro basarsi appunto sulla equazione ricorsiva di Bellman: la gran parte dei metodi TD hanno un tal parametro λ, $0 \leq \lambda \leq 1$, che è in grado di interpolare continuamente tra i metodi Monte Carlo, che non si basano su equazioni di Bellman, e modelli TD di base che vi si basano completamente. Questo può essere efficace nel mitigare il problema.

2.4.6 Ricerca diretta della policy (*Direct policy search*)

Un metodo alternativo è quello di cercare direttamente in (un qualche sottoinsieme) dello spazio delle politiche, nel qual caso il problema diventa un caso di ottimizzazione stocastica (*stochastic optimization*).

I due approcci disponibili sono i metodi basati sul gradiente e quelli liberi (*gradient free*).

I metodi basati sul gradiente (*policy gradient methods*) cominciano con una mappatura da uno spazio finito-dimensionale (parametrico) allo spazio delle politiche: dato il vettore parametro θ, sia π_θ la politica ad esso associata. Definendo la funzione di performance come $\rho(\theta) = \rho^{\pi_\theta}$, sotto alcune condizioni (deboli) questa funzione sarà differenziabile come funzione del vettore parametro θ.

Se il gradiente di ρ fosse noto, si potrebbe usare l'*ascesa del gradiente.* Poiché un'espressione analitica del gradiente non è disponibile, si ha solo una stima affetta da rumore; una tale stima può essere costruita in vari modi, il che dà origine ad algoritmi come il metodo REINFORCE di Williams[14] (che è noto conosciuto come il metodo del rapporto di probabilità nella letteratura dell'ottimizzazione basata su simulazione).[15]

Molti dei metodi di ricerca della politica potrebbero rimanere bloccati in ottimi locali, dato che sono basati su ricerca *locale*.

Una grande classe di metodi evita di basarsi su informazioni sul gradiente; questi includono: la ricottura simulata (*simulated annealing*), la ricerca dell'entropia incrociata (*cross-entropy search*), o metodi di computazione evolutiva (*evolutionary computation*).

Molti metodi *gradient-free* sono in grado di raggiungere (nella teoria e nel limite) un ottimo globale.

---

[13] Watkins, Christopher J.C.H. (1989). *Learning from Delayed Rewards* (PDF) (PhD thesis). King's College, Cambridge, UK.

[14] Williams, Ronald J. (1987). "A class of gradient-estimating algorithms for reinforcement learning in neural networks". *Proceedings of the IEEE First International Conference on Neural Networks.* CiteSeerX 10.1.1.129.8871.

[15] Peters, Jan; Vijayakumar, Sethu, Stefan (2003). "Reinforcement Learning for Humanoid Robotics" (PDF). IEEE-RAS International Conference on Humanoid Robots.

I metodi di ricerca della politica potrebbero convergere lentamente in caso di dati affetti da rumore. Ad esempio, questo accade in problemi episodici quando le traiettorie sono lunghe e la varianza dei ritorni è grande. In questo caso potrebbero aiutare i metodi basati sulla value function che si affidano alle differenze temporali.
In anni recenti, sono stati proposti metodi 'actor-critic' i quali hanno rivelato buona performance in diversi problemi. [16]

Sia il comportamento asintotico che quello da campione finito della maggior parte degli algoritmi sono ben compresi, e algoritmi con probabile buona performance online (affrontando il problema dell'esplorazione) sono noti.

Un'efficiente esplorazione dei MDP è data in Burnetas e Katehakis (1997)[8]. I limiti della performance a tempo finite sono anch'essi apparsi per molti algoritmi, ma ci si aspetta che siano piuttosto blandi, per cui è necessario maggior lavoro per capire meglio i relativi vantaggi e limitazioni.

Per gli algoritmi incrementali, sono stati posti problemi di convergenza asintotica.
Gli algoritmi basati sulle differenze temporali convergono sotto un insieme di condizioni più ampio di quanto fosse possibile in precedenza: per esempio, quando usati con un'arbitraria approssimazione ad una funzione liscia.

2.4.7 Comparazione degli algoritmi per il RL

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Model-Free	Either	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	Model-Free	On-policy	Discrete	Discrete	Q-value

[16] Juliani, Arthur (2016-12-17). "Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C)". *Medium.* Retrieved 2018-02-22.

Q-learning - Lambda	State–action–reward–state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	Model-Free	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Model-Free	Off-policy	Continuous	Continuous	Advantage

TABELLA 2.1 COMPARAZIONE DEGLI ALGORITMI PER IL RL

2.5 Note conclusive

In base a quanto visto, possiamo far notare che il RL è particolarmente adatto per problemi che includono un trade off di tipo 'breve termine' contro 'lungo termine'; è stato applicato con successo a vari problemi, tra cui il controllo robotico, schedulazione dei montacarichi, telecomunicazioni, backgammon, checkers [17] e Go (AlphaGo).

[17] Sutton & Barto, 1998. Capitolo 11

Capitolo 3: Il sistema in esame

Questo capitolo è dedicato al sistema reale che prendiamo in esame e comprende la sua descrizione, caratteristiche e processo.

3.1 Descrizione del sistema

Il sistema che consideriamo è quello relativo ad un'azienda polacca produttrice di un pallet metallico.

Parliamo di una piccola azienda, considerata un business di famiglia, che produce un pallet metallico come mostrato in figura 3.1, per cui un prodotto grande, ingombrante, e la cui produzione risulta abbastanza complessa. La produzione è essenzialmente manuale; l'azienda produce solamente il prodotto in questione ed impiega 10 operatori, di cui 9 lavoratori (*workers*) generici parimenti specializzati ed un saldatore, il quale possiede delle capacità (*skills*) differenti e si deve occupare di tutte le attività di saldatura necessarie. Anche il trasporto del prodotto in lavorazione (WIP) e dei materiali è manuale: non vi sono veicoli a motore che supportino i flussi di materiali o di WIP all'interno del processo.

FIGURA 3.1 PALLET METALLICO

L'azienda è organizzata in stazioni di lavoro, ognuna delle quali può eseguire uno o più *task* (attività), e il cui elenco con relativa descrizione è dato in tabella 3.1.

Il layout dello stabilimento non è ottimizzato, ma poiché si trova all'interno di un edificio vecchio, con muri strutturali tra le stazioni, non è possibile agire sul layout e lo si prende per dato; il layout dello stabilimento è presentato in figura 3.2.

Stazione	**Operazione effettuata**	**Descrizione**
MS-1	Magazzino profili	Deposito per la materia prima 'profilo'
MS-2	Magazzino materiale grezzo	Deposito per le materie prime 'lastre di metallo' e 'angoli di metallo'
st 2	Taglio lastre	Controllo e determinazione delle dimensioni delle lastre metalliche
st 3	Taglio degli angoli delle lastre	Aggiustamento degli angoli delle lastre metalliche
st 4	Piegatura lastre	Piegatura delle lastre di metallo
st 5	Saldatura	Tutte le attività di saldatura: della coppa, cornice, lati e fondo.
st 6	Taglio profilo	Controllo e determinazione delle dimensioni dei profili laterali
st 7	Incisione profilo	Incisione dei buchi laterali
st 8	Perforazione buchi	Perforazione dei buchi a cui altre parti verranno assemblate
st 9	Taglio angoli	Aggiustamento degli angoli
st 10	Consolidazione e assemblaggio	Assemblamento finale del prodotto finito
st 11	Consolidazione e assemblaggio	Assemblamento finale del prodotto finito
PS	Magazzino prodotti pronti	Deposito del prodotto finito

TABELLA 3.1 STAZIONI DI LAVORO

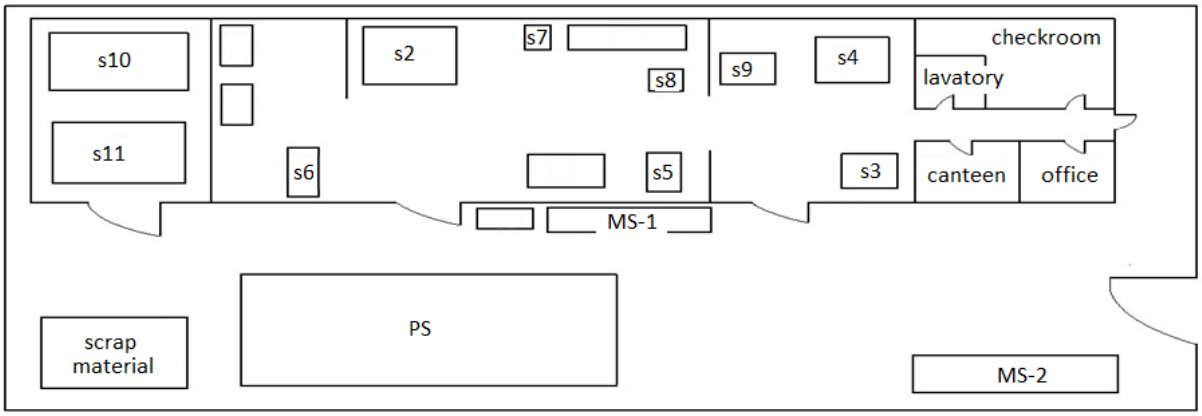


FIGURA 3.2 LAYOUT DELLO STABILIMENTO

La produzione implica 13 attività, al di là delle attività di trasporto. Ciascuna attività può necessitare di uno o di due operatori, le cui abilità (*skills*) determinano il fatto che uno possa effettuare una certa attività o meno; vale a dire, non tutti gli operatori possono effettuare tutti i task.

Le attività di manifattura, insieme alle durate, alle corrispondenti stazioni di lavorazione e al numero di operatori impiegati sono presentate in tabella 3.2.

Attività manufatturiera (*mt*)	**Descrizione**	**Durata**	**Dimensione del lotto**	**Stazione in cui l'attività può essere eseguita**	**Numero di operatori necessari per l'esecuzione**
mt 1	Taglio lastra	40	1	*st 2*	2
mt 2	Taglio angoli della lastra	652	1	*st 3*	2
mt 3	Piegatura lastra	624	1	*st 4*	2
mt 4	Taglio profilo	349	4	*st 6*	1
mt 5	Incisione profilo	504	4	*st 7*	1
mt 6	Perforazione buchi	1026	4	*st 8*	1
mt 7	Taglio degli angoli	16	4	*st 9*	1
mt 8	Saldatura coppa	192	-	*st 5*	1
mt 9	Saldatura angolo	304	-	*st 5*	1
mt 10	Saldatura profilo	416	-	*st 5*	1
mt 11	Saldatura fondi	120	-	*st 5*	1
mt 12	Consolidazione	1187	1	*st 10, st 11*	2
mt 13	Assemblaggio	1212	1	*st 10, st 11*	2

TABELLA 3.2 ATTIVITÀ MANUFATTURIERE

È importante sottolineare che le attività di trasporto necessarie per passare da un'attività ad un'altra vengono effettuate a lotti costituiti dal numero esatto di pezzi necessari per produrre UNA unità del prodotto finale. Perciò, poiché servono 4 profili per produrre un pallet, gli operatori aspettano finché 4 profili siano tagliati (*mt 4*) per spostarli all'attività di incisione (*mt 5*), dove allo stesso modo verranno lavorati in sequenza e poi trasportati alla tappa successiva come un lotto.

Lo stesso principio si applica a tutti i pezzi e componenti; dunque, poiché tutti i pezzi di un lotto vengono lavorati in serie prima che il risultato (output) di un'attività sia trasportato, la durata di ciascuna attività presentata in tabella 3.2 è il tempo aggregato necessario per processare tutti gli oggetti che formano un lotto.

Ad esempio, una lastra costituisce l'output di *mt 1* (taglio lastra), per cui 40s è il tempo necessario per caricare, processare e scaricare un foglio. Diversamente, un lotto in *mt 7* (taglio degli angoli) è fatto di 4 pezzi, per cui 16 s, -la durata riportata nella tabella-, è il tempo necessario per caricare, processare e scaricare tutti e 4 gli angoli, il che implica che 4s è il tempo necessario per lavorare un singolo angolo (16s/4unità=4s).

Per illustrare meglio il processo di produzione, la figura 3.3 presenta uno schema della BOM (Bill of Materials) multilivello del prodotto.

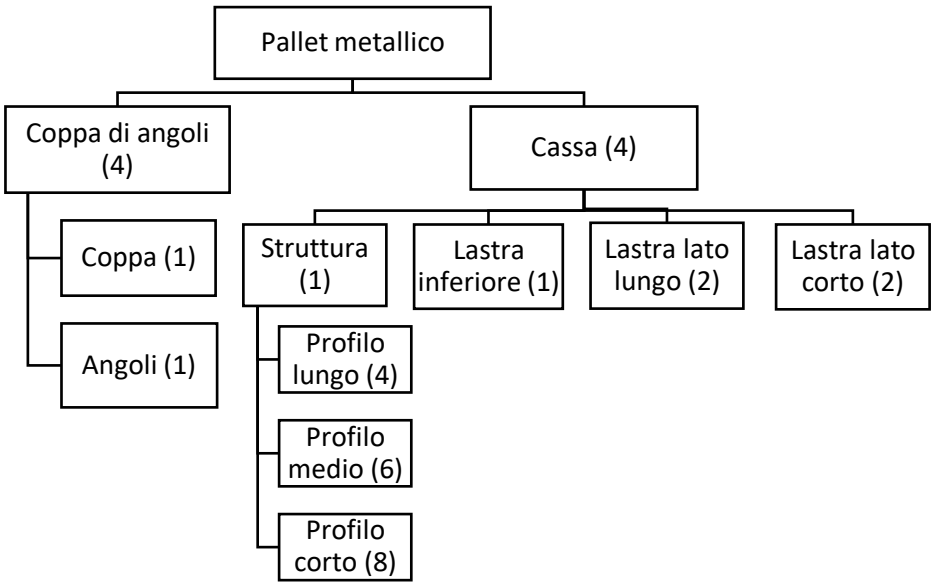


FIGURA 3.3 BILL OF MATERIALS

Come accennato, fra due attività manufatturiere vi è un trasporto, che viene effettuato manualmente da uno o più operatori: si porta il lotto di oggetti semi finiti da una stazione di lavoro alla successiva, secondo la sequenza di lavoro. Questo tempo di trasporto richiesto non include le operazioni di carico e scarico, che abbiamo visto essere già incluse nella durata dell'attività.

Come facilmente intuibile, la produzione richiede anche delle attività di trasporto delle materie prime (ad inizio processo) e dei prodotti finiti (a fine processo) verso i magazzini dedicati, per cui il processo include anche tali attività di trasporto.

I tempi necessari per le attività di trasporto sono stati misurati in azienda, e sono riportati nella successiva tabella 3.3, insieme alle relazioni di precedenza e successione dei flussi interni di materiali e WIP e al numero di operatori necessari:

Attività di trasporto (*tt*)	**Stazione precedente**	**Carico trasportato**	**Durata [s]**	**Stazione successiva**	**Numero di operatori necessari per l'esecuzione dell'attività**
tt 1	*MS-2*	Lastra	20	*st 2*	1
tt 2	*st 2*	Lastra tagliata	20	*st 3*	2
tt 3	*st 2*	Lastra tagliata	15	*st 5*	1
tt 4	*st 3*	Lastra senza angoli	5	*st 4*	2
tt 5	*st 4*	Lastra piegata	25	*st 10*	2
tt 6	*st 4*	Lastra piegata	25	*st 11*	2
tt 7	*MS-1*	Profili	10	*st 6*	1
tt 8	*st 6*	Profili tagliati	15	*st 7*	1
tt 9	*st 7*	Profili incisi	5	*st 8*	1
tt 10	*st 8*	Profili coi buchi	10	*st 5*	1
tt 11	*MS-2*	Angoli	15	*st 9*	1
tt 12	*st 9*	Angoli tagliati	10	*st 5*	1

tt 13	*st 5*	Cornice	20	*st 10*	2
tt 14	*st 5*	Cornice	20	*st 11*	2
tt 15	*st 5*	Coppe	15	*st 10*	1
tt 16	*st 5*	Coppe	15	*st 11*	1
tt 17	*st 10*	Trasporto del pallet	15	*PS*	2
tt 18	*st 11*	Trasporto del pallet	15	*PS*	2

TABELLA 3.3 ATTIVITÀ DI TRASPORTO

Per quanto riguarda le competenze degli operatori, le attività nel complesso sono basiche, e tutti gli operatori sono in grado di eseguirle. L'unica eccezione è costituita dalle attività di saldatura, che avvengono nella stazione S5: la saldatura richiede un certo livello di abilità, che possiede solo un operatore (il saldatore), il quale rimane e rimarrà fisso nella stazione S5. Tutte le altre attività manufatturiere, così come le attività di trasporto, sono eseguite dagli altri operatori.

Grazie a tutti i dati e le informazioni sul processo che abbiamo, possiamo costruire il diagramma di flusso del processo, che mostra come procede la produzione, -con le relative precedenze-, dall'inizio fino alla disponibilità del prodotto (pallet) finito, ossia la *sequenza delle attività*. Tutto ciò è riportato in figura 3.4, dove, con i rettangoli bianchi sono in indicati tutti i manufacturing task, nell'ordine in cui devono essere eseguiti, i rettangoli arancioni sono i depositi. Le frecce indicano tutte che vi è una relazione di precedenza tra i rispettivi task (o allo stesso modo, che l'output di un'attività è input di un'altra), ma quelle nere stanno ad indicare che vi è necessità di uno spostamento fisico di materiale (ergo sarà necessaria una o più attività di trasporto preliminare) mentre quelle celesti indicano che non vi è necessità di trasporto di materiale per passare dall'attività precedente a quella successiva: in altri termini, l'attività successiva viene eseguita nella medesima stazione. La freccia in uscita dal deposito dei prodotti finiti (PS) è stata inserita solo per indicare che vi è un flusso in uscita dei prodotti finiti dal relativo magazzino, ma questo trasporto non interverrà nei nostri calcoli e ragionamenti: non abbiamo alcuna attività di trasporto dal magazzino PS in fuori in tabella 3.3, la nostra analisi si conclude con il deposito del pallet nel magazzino.

Dal diagramma possiamo vedere che vi sono 4 percorsi di processamento, dove la fine di ciascun percorso è la medesima, in quanto i percorsi si uniscono tutti in uno unico prima delle operazioni di consolidamento e assemblaggio.

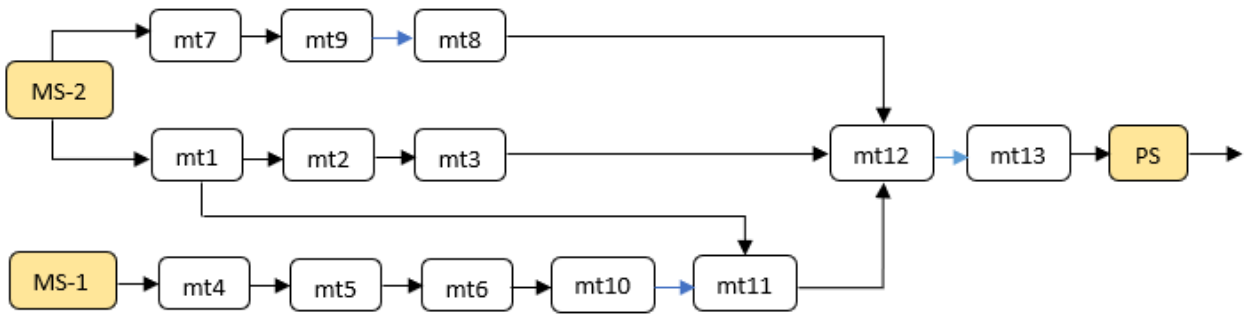


FIGURA 3.4 SEQUENZA ATTIVITÀ

Nello studiare un processo, è sempre buona cosa evidenziare il percorso critico del diagramma. Il cammino (o percorso) critico di un sistema è un concetto fondamentale in ingegneria gestionale, ed in particolare nella gestione della produzione / dei processi perché determina il tempo di flusso (*flow time,* FT) dell'intero processo: è il percorso con la durata (intesa come somma delle durate delle singole attività che lo compongono) maggiore, e questo è il motivo per cui determina FT.

Verificando le durate dei percorsi, si trova che il percorso critico è l'ultimo, costituito dalle attività sequenziali mt4, mt5, mt6, mt10, mt11, mt12, e mt13, con una durata di 4814 secondi.

Le attività di consolidamento e assemblaggio, -mt12 ed mt13-, che hanno le durate più lunghe, possono essere effettuate sia sulla stazione 10 che sulla 11, che dunque costituiscono un parallelo di due macchine.

Nella successiva figura 3.6 troviamo un ulteriore schema del sistema, in cui il flusso di lavoro è evidenziato assieme alla rappresentazione delle stazioni (di lavoro e di magazzino) esistenti e disponibili per l'esecuzione dei task relativi, alle attività di trasporto necessarie, e alla quantità di operatori richiesti dai *manufacturing task* (non è rappresentata la quantità di operatori richiesta per i task di trasporto, per la quale si rimanda alla tabella 3.3.

Da questa figura, è evidente la configurazione in parallelo delle stazioni *st*10 e *st*11.

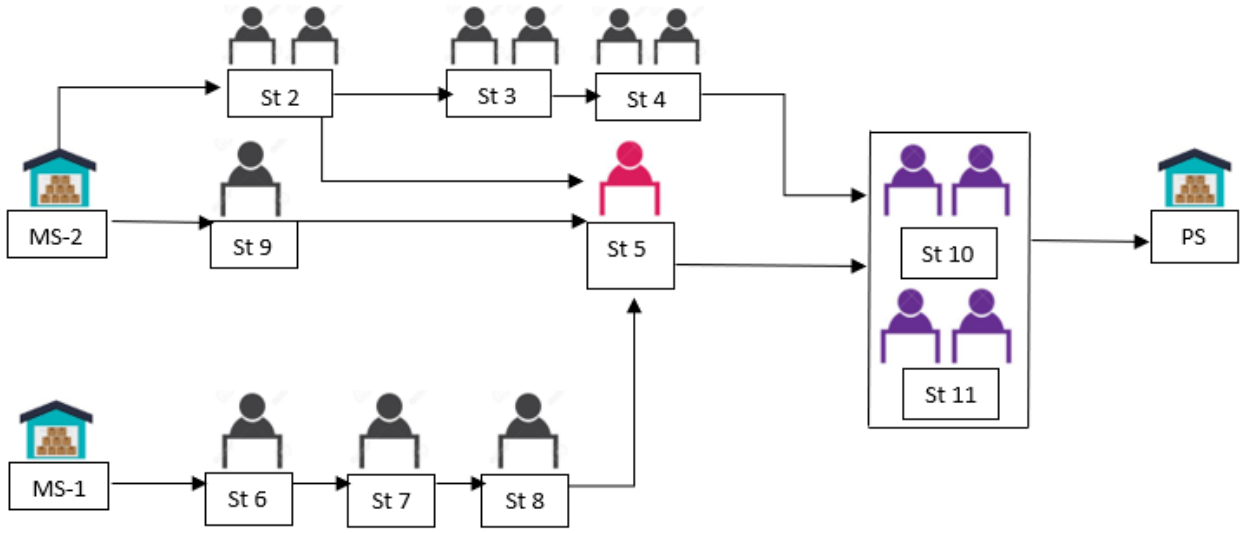


FIGURA 3.6 RAPPRESENTAZIONE DEL FLUSSO DI LAVORO

3.2 Dettagli e ipotesi del processo

Teniamo sempre presente la distribuzione delle risorse nello stabilimento (il layout), come da fig. 3.2; la porzione di stabilimento a destra dell'ingresso è dedicata alle aree funzionali alle risorse umane (la toilette, l'area ristoro, l'ufficio e la mensa), nelle quali tutti i dipendenti dell'azienda lavorano o spendono parte del proprio tempo (ad esempio in caso di pause e/necessità); tuttavia, per quanto riguarda il lavoro degli operai, -ciò che (come spiegheremo meglio in seguito) ci interessa allocare al meglio e controllare-, supponiamo che essi partano (prima di essere assegnati), -e vi ritornino, in caso abbiano finito una

lavorazione e non ne abbiano una successiva da effettuare subito dopo-, da un 'punto di ritrovo' che immaginiamo trovarsi in un posto comodo e funzionale: all'incirca al centro, in basso rispetto alle stazioni e ai magazzini, in modo tale che non vi siano stazioni e magazzini privilegiati o svantaggiati in termini di raggiungimento e quindi di distanze percorse dagli operai. Questo discorso vale per gli operai semplici, mentre per quanto riguarda il saldatore, egli sarà posizionato di base, per il lavoro, nella stazione *St5*, nella quale avvengono tutte quante le operazioni di saldatura.

Gli spostamenti del saldatore saranno molto limitati, e per di più non si occuperà delle attività di trasporto: tutte le attività di trasporto sono a carico degli operatori semplici, i quali svolgeranno i task precedenti e successivi a quelli di saldatura e le relative attività di trasporto.
Il saldatore effettua tutte le operazioni che devono essere effettuate sulla stazione 5, rispettando le precedenze delle attività di saldatura, in modo da non ritardare il processo. La stazione 5 si trova sul percorso critico; anche per questo motivo, ad egli non vengono assegnate attività extra: si occupa strettamente del lavoro di saldatura.

Per quanto riguarda i parametri del processo in ogni singola stazione, i tempi di processamento (le durate dei singoli task) seguono una distribuzione triangolare simmetrica (distribuzione di Simpson) con moda (coincidente in questo caso con la media) data dai valori deterministici specificati nella tabella 3.2 relativa alle attività manufatturiere, e scostamenti pari a $\mp$10%.
Per quanto riguarda le variabili relative ai guasti e setup delle macchine (ad esempio Mean Time To Repair-MTTF o Mean Time To Failure, Setup Time), queste restano fuori dallo scopo del lavoro. Nonostante questa assunzione possa sembrare irrealistica, in realtà non lo è perché il processo in esame è un processo molto tradizionale e poco innovativo: vi è una grandissima influenza del lavoro manuale degli operai, il processo non è particolarmente tecnologizzato. Le macchine utilizzate sono per lo più manuali, ausili al lavoro degli operatori; la frequenza dei guasti è irrilevante e la riparazione/sostituzione molto facile e veloce. Allo stesso modo, i tempi di setup delle macchine possono essere considerati nulli dal momento in cui non sono previsti/necessari cambi di utensili, avvii preventivi, tempi di attesa ecc.

Dunque, - in pratica-, gli unici tempi in cui il sistema può eventualmente bloccarsi (in toto o in qualche parte) sono quelli in cui non vi è un operatore disponibile per effettuare il task richiesto, nel rispetto delle precedenze.

Ciò che ci interessa controllare in questo processo, e su cui vorremmo andare poi a valutare l'apprendimento per rinforzo, è l'allocazione delle risorse umane, ossia degli operatori.

Vogliamo anche studiarlo, - e per farlo ci avvarremo della metodologia della simulazione ad eventi discreti-, il numero degli operatori.

L'ipotesi che facciamo è quella di voler allocare gli operatori minimizzando la distanza percorsa, in quanto in un precedente studio (riferimento n.4 della bibliografia) del sistema, nel quale è stata fatta la medesima ipotesi di presenza di un '*dispatcher*'/punto di ritrovo intermedio da cui far partire tutti gli operatori, sono state comparate le 4 politiche di allocazione delle risorse:

1. *Round Robin*, secondo la quale il task viene allocato all'operatore in base alla sequenza ordinale degli operatori (le assegnazioni seguono una rotazione);

2. *First Available*, secondo la quale viene assegnato il primo operatore disponibile;

3. *Shortest Distance if Available*, secondo la quale viene assegnato l'operatore più vicino tra quelli disponibili;

4. *Shortest Queue if Available*, secondo la quale viene assegnato l'operatore con la minore sequenza di lavoro assegnata in coda, e che in caso di nessun operatore disponibile, crea una coda basandosi sulla minor distanza;

ed è stato stabilito che la politica in questione, -cioè la 3-, fosse la migliore.

La motivazione di ciò sta, -secondo lo studio -, nel fatto che, seppur non vi sia differenza rispetto agli altri metodi di allocazione,- 2 e 4 -, per quanto riguarda il tasso di produzione (TH) o il tempo ciclo (CT), vi sia per quanto riguarda il bilanciamento del carico di lavoro degli operatori, con il metodo 3. che dà i migliori risultati in tal senso.

Il metodo 1 invece, secondo lo studio è il peggiore, con una minore performance anche di sistema.

Per questo motivo, assumeremo nel nostro modello di sistema iniziale, -che esplichiamo nel capitolo successivo-, di scegliere come metodo di allocazione per gli operatori (standard) *Shortest Distance* (if *Available* nel nostro caso è implicito).

Un'altra ipotesi che facciamo sul nostro sistema di partenza è: supponiamo di analizzare dapprima il sistema nella condizione reale, quindi con 9 operatori standard impiegati e 1 saldatore, e che sia consentito un numero massimo di work in progress di 4 (dunque un numero massimo consentito di pezzi in processamento nel sistema in contemporanea di 5).

Il modello iniziale del sistema sarà stilato su queste basi.

Capitolo 4: Modellizzazione del sistema mediante codice Matlab

Questo capitolo spiega la modellizzazione del sistema che è stata eseguita e il modello stesso.

Il modello in questione è allegato alla tesi (Allegato 1); si tratta di un file in formato *.m*, ossia da leggere, ed eventualmente far girare, sul software *Matlab*. Lo script contiene già in sé una grande quantità di commenti, - riconoscibili perché preceduti dal simbolo di percentuale ed automaticamente colorati dal programma in verde -, a spiegazione delle variabili definite e dello scopo e funzionamento del modello, che altrimenti sarebbero complessi da capire semplicemente leggendo. I commenti risultano utili non solamente per terzi che si trovano a leggere o a dover utilizzare il modello, ma anche per me che l'ho fatto, sia a lavoro concluso, che mentre lo redigevo.

Dunque, nell'Allegato A è possibile vedere il modello nella sua interezza, corredato di numerosi commenti; nonostante ciò, spieghiamo le parti fondamentali del modello e del suo funzionamento nel seguito, - senza entrare in tutti i dettagli, per i quali si rimanda all'allegato-.

La modellizzazione in questione assume di voler modellizzare il sistema nella sua configurazione di partenza, ossia quella impiegata dall'azienda, che dispone 9 operatori standard e un saldatore.

4.1 Rappresentazione dei dati di sistema

Per rappresentare il sistema in codice, abbiamo dovuto effettuare delle modifiche nella rappresentazione dei dati del sistema, che per chiarezza del lavoro nel complesso riportiamo, di seguito, insieme ad eventuali specificazioni e rappresentazioni del sistema ulteriori a quelle già fatte nel capitolo 3.

- La successiva tabella 4.1 riporta le corrispondenze tra i codici che abbiamo assegnato in Matlab alle stazioni e i nomi delle stazioni originali (quelli della tabella 3.1), insieme alla specificazione dei manufacturing task che possono essere eseguiti su ciascuna stazione.
È stata aggiunta una stazione al fondo, a rappresentare il 'punto di ritrovo' di cui abbiamo detto prima, che abbiamo chiamato *Dispatcher*, ad indicare che è il luogo da cui vengono 'dispacciati' gli operatori.

Stazione	Equivalente nome reale	MT (manufacturing task)
1	MS-1	
2	MS-2	
3	St2	1
4	St3	2
5	St4	3

6	St5	8
		9
		10
		11
7	St6	4
8	St7	5
9	St8	6
10	St9	7
11	St11	12,13
12	St10	12,13
13	PS	
14	Dispatcher	

TABELLA 4.1 RAPPRESENTAZIONE MATLAB DELLE STAZIONI

- La successiva tabella 4.2 riporta un aggregato di dati utili sui manufacturing task, oltre alla separazione dei manufacturing task 12 e 13, che possono essere eseguiti indifferentemente sulla stazione 11 o sulla stazione 12, - come mostrato anche nella tabella stessa-, in due task distinti, in modo tale che possano essere trattati separatamente e che si sia in grado, nel codice, di assegnare una stazione o l'altra ad un certo job, e allo stesso tempo di risalire a quale delle due stazioni è stata assegnata.

La separazione (*split*) è necessaria anche per modellare il fatto che se viene scelto di effettuare la lavorazione di un dato job diciamo nella stazione 11, il percorso seguito nella parte finale del processo (ossia dalla consolidazione e assemblaggio al rilascio) sarà mt12a, mt13a, tt17, ossia non è possibile trasferire il job sulla stazione parallela per la lavorazione successiva.

MT	Stazione	Num. operatori necessari	tempo min di processamento	tempo medio di processamento	tempo max di processamento	Eventuale TT predecessore	Eventuale TT successore
1	3	2	36	40	44	1	2,3
2	4	2	586,8	652	717,2	2	4
3	5	2	561,6	624	686,4	4	5,6
4	7	1	314,1	349	383,9	7	8
5	8	1	453,6	504	554,4	8	9
6	9	1	923,4	1026	1128,6	9	10
7	10	1	14,4	16	17,6	11	12
8	6	1	172,8	192	211,2	-	15,16
9	6	1	273,6	304	334,4	12	-
10	6	1	374,4	416	457,6	10	-
11	6	1	108	120	132	3	13,14
12	11/ 12	2	1068,3	1187	1305,7	15,16,5,6,13,14	-
13	11/ 12	2	1090,8	1212	1333,2	-	17,18
12a	11	2	1068,3	1187	1305,7	15,5,13	-
12b	12	2	1068,3	1187	1305,7	16,6,14	-
13a	11	2	1090,8	1212	1333,2	-	17
13b	12	2	1090,8	1212	1333,2	-	18

TABELLA 4.2 DATI AGGREGATI SUI MANUFACTURING TASK E SPLIT TASK DI CONSOLIDAMENTO E ASSEMBLAGGIO

Dunque, vediamo come abbiamo 'splittato' il task 12 nei task 12a e 12b, rispettivamente eseguibili in stazione 11 e 12, e il 13 in 13a e 13b, sempre rispettivamente eseguibili nelle stazione 11 e 12, che come abbiamo detto precedentemente, costituiscono un parallelo di due macchine identiche.

A seconda della stazione scelta per il consolidamento e assemblaggio, il task di trasporto che serve per spostare il prodotto finito da lì al magazzino dei prodotti finiti (PS) sarà tt17 nel caso di scelta della stazione 11 e tt18 in caso invece di scelta della stazione 12.

La tabella che segue riporta la denominazione aggregata assegnata ai task per Matlab: i task sono numerati in modo crescente, ponendo prima i manufacturing task (con lo split di cui poco sopra) e successivamente i *transport task*.

Equivalente codice	Task
MT1	1
MT2	2
MT3	3
MT4	4
MT5	5
MT6	6
MT7	7
MT8	8
MT9	9
MT10	10
MT11	11
MT12a	12
MT12b	13
MT13a	14
MT13b	15
TT1	16
TT2	17
TT3	18
TT4	19
TT5	20
TT6	21
TT7	22
TT8	23
TT9	24
TT10	25
TT11	26
TT12	27
TT13	28
TT14	29
TT15	30
TT16	31
TT17	32

TT18	33

TABELLA 4.3 CORRISPONDENZA TRA MT, TT, E TASK MATLAB

- Le successive tabelle 4.4 - 4.7 sono tutte relative ai percorsi, coerentemente con la descrizione del sistema precedentemente.

La tabella 4.4 riporta i percorsi che costituiscono il sistema: ivi sono indicati i nomi dei task ed anche le precedenze, per ogni percorso.

Percorsi												
TT7	MT4	TT8	MT5	TT9	MT6	TT10	MT10	MT11	TT13	MT12a	MT13a	TT17
TT1	MT1	TT3							TT14			
		TT2	MT2	TT4	MT3	TT5						
						TT6				MT12b	MT13b	TT18
TT11	MT7	TT12	MT9	MT8	TT15							
					TT16							

Devo assegnarle entrambe una volta finita MT1

Ne assegno una, di base la prima. Se vi è un wip *in need*, posso assegnare il task gemello.

TABELLA 4.4 PERCORSI

La tabella 4.5 riporta allo stesso modo i percorsi e le precedenze, ma la rappresentazione è un po' diversa ed è con codici Matlab anziché con i nomi dei task.

Percorsi													
1	26	7	27	9	8	31					13	15	33
						30					12	14	32
2	16	1	17	2	19	3	21				13	15	33
							20				12	14	32
3				18					11	29	13	15	33
4	22	4	23	5	24	6	25	10		28	12	14	32

TABELLA 4.5 PERCORSI CON CODICI MATLAB

Le successive tabelle 4.6 e 4.7 sono progettate in vista della progettazione del codice del sistema.

La tabella 4.6 riporta la rappresentazione statica dei percorsi, in cui la parte finale dei percorsi (dall'assemblaggio e consolidamento in poi), comune a tutti, viene indicata solo una volta, in

corrispondenza del percorso 1, che è il più lungo (assieme al suo gemello 2). In ultima colonna è indicata la durata complessiva dei task presenti in ciascun percorso.

Indice percorso Matlab	MATRICE STATICA													Durata media
1	26	7	27	9	8	31								552
2						30								15
3	16	1	17	2	19	3	21							1386
4							20							25
5			18											15
6	22	4	23	5	24	6	25	10	11	29	13	15	33	4889
7										28	12	14	32	2434

TABELLA 4.6 MATRICE STATICA DEI PERCORSI MATLAB

La tabella 4.7 è analoga alla precedente, con la sola differenza che le posizioni non vuote sono sostituite dal valore 1 e quelle vuote dal valore 0, ad indicare la presenza o meno di un task in una certa posizione.
Questa matrice è pensata per essere dinamica, ossia con i valori (0 e 1) che si modificano con l'evolversi del sistema nel tempo.

Indice percorso Matlab	MATRICE DINAMICA													Durata media
1	1	1	1	1	1	1	0	0	0	0	0	0	0	552
2	0	0	0	0	0	1	0	0	0	0	0	0	0	15
3	1	1	1	1	1	1	1	0	0	0	0	0	0	1386
4	0	0	0	0	0	0	1	0	0	0	0	0	0	25
5	0	0	1	0	0	0	0	0	0	0	0	0	0	15
6	1	1	1	1	1	1	1	1	1	1	1	1	1	4889
7	0	0	0	0	0	0	0	0	0	1	1	1	1	2434

TABELLA 4.7 MATRICE DINAMICA DEI PERCORSI MATLAB

4.2 Il modello del processo

4.2.1 Definizione delle variabili

Il modello del processo incomincia con la sezione dedicata alla definizione delle variabili. In particolare, troviamo:

- *out*, che memorizzerà gli istanti di tempo in cui ciascun job in sequenza viene completato e rilasciato;

- le variabili *W*, *Wt*, e *Wp*, tutte relative agli operatori impiegati in produzione e indicanti rispettivamente:

  - *W*, se l'operatore in dato stato risulta libero o occupato (variabile booleana)

- *Wt*, in corrispondenza degli '1' della variabile precedente, specifica con quale task dato operatore risulta occupato
- *Wp* invece, non contiene zeri ed indica la posizione che ciascun operatore occupa in ciascuno stato, quindi tiene traccia degli spostamenti degli operatori (standard).

In queste variabili, le posizioni da 1 a 9 si riferiscono agli operatori 1-9 (ciascuno identificato univocamente dal suo numero), e l'ultima (posizione 10) al saldatore. Per l'ipotesi fatta sul saldatore, che non si sposta dalla stazione 6 di saldatura e non effettua trasporti, il valore in posizione 10 della variabile *Wp* sarà sempre 6, per ogni stato.

- *incorso* segna, per ogni stato in cui si troverà il sistema nella sua evoluzione, quali task sono in corso: serve per controllare l'output della simulazione, una volta ottenuto.
- *produzione*: viene incrementata di 1 ogni volta che una nuova unità di prodotto è completata, per cui parte dal valore zero e può essere volendo utilizzata come valore target per terminare la simulazione.
- le variabili di tipo *perc…* servono per segnare, per ciascun prodotto in corso (che può essere il prodotto in lavorazione principale o un Work In Progress) se per la parte finale del processo è stato assegnato il percorso del parallelo che utilizza la stazione 11, oppure quello che utilizza la stazione 12.
- la variabile *St*, - a valori booleani -, segna, per ogni stato toccato, se una determinata stazione risulta già occupata, in modo che non possa esservi assegnato un altro task da eseguirvi sopra.
- la variabile *S*, molto simile alla precedente e anch'essa a valori booleani, assegna il valore booleano 1 alle stazioni per le quali è stato nello step corrente già identificato un task da assegnare che necessita di quella stazione, in modo da non trovarsi con due task da assegnare che competono per la stessa macchina.

Le successive variabili centrali che spieghiamo sono fondamentali:

- *dist* è la matrice delle distanze tra le stazioni. È una matrice statica, che servirà per poter confrontare la distanza che andrebbero a percorrere gli operatori che competono per l'assegnazione di un dato task in un dato istante temporale.
- *task* è la matrice che riporta i dati necessari alla codificazione, relativi a tutti i 33 task. Spiegazioni circa le sue colonne sono date nel codice (sempre All.1) e, poiché si tratta di una matrice di fondamentale importanza per la comprensione del codice, è riportata, non in codice, di seguito (tabella 4.8). La tabella, poiché è molto ampia, è stata spezzata in due parti, in modo da poter essere letta agevolmente.

Equivalente codice	Task	Numero operatori richiesti	Precedenza				Successione			Stazione di lavoro / di partenza
MT1	1	2	16				17	18	1*	3
MT2	2	2	17				19			4
MT3	3	2	19				21	20	2*	5
MT4	4	1	22				23			7
MT5	5	1	23				24			8
MT6	6	1	24				25			9

MT7	7	1	26				27			10
MT8	8	1	9				31	30	2	6
MT9	9	1	27				8			6
MT10	10	1	25				11			6
MT11	11	1	18	10		1*	29	28	2	6
MT12a	12	2	30	20	28	1	14			11
MT12b	13	2	31	21	29	1	15			12
MT13a	14	2	12				32			11
MT13b	15	2	13				33			12
TT1	16	1	-				1			2
TT2	17	2	1				2			3
TT3	18	1	1				11			3
TT4	19	2	2				3			4
TT5	20	2	3				12			5
TT6	21	2	3				13			5
TT7	22	1	-				4			1
TT8	23	1	4				5			7
TT9	24	1	5				6			8
TT10	25	1	6				10			9
TT11	26	1	-				7			2
TT12	27	1	7				9			10
TT13	28	2	11				12			6
TT14	29	2	11				13			6
TT15	30	1	8				12			6
TT16	31	1	8				13			6
TT17	32	2	14				-			11
TT18	33	2	15				-			12

* 1 indica che entrambe le attività succeessive devono a prescindere essere eseguite. 2 significa che devono essere eseguite secondo necessità (almeno una delle due sempre)

Stazione di arrivo	tempo min di processamento	tempo medio di processamento	tempo max di processamento
3	36	40	44
4	586,8	652	717,2
5	561,6	624	686,4
7	314.1	349	383.9
8	453.6	504	554.4
9	923.4	1026	1128.6
10	14.4	16	17.6
6	172.8	192	211.2
6	273.6	304	334.4
6	374.4	416	457.6

6	108	120	132
11	1068,3	1187	1305,7
12	1068,3	1187	1305,7
11	1090,8	1212	1333,2
12	1090,8	1212	1333,2
3	18	20	22
4	18	20	22
6	13,5	15	16,5
5	4,5	5	5,5
11	22,5	25	27,5
12	22,5	25	27,5
7	9	10	11
8	13,5	15	16,5
9	4,5	5	5,5
6	9	10	11
10	13,5	15	16,5
6	9	10	11
11	18	20	22
12	18	20	22
11	13,5	15	16,5
12	13,5	15	16,5
13	13,5	15	16,5
13	13,5	15	16,5

TABELLA 4.8 TABELLA DEI TASK MATLAB

- *P* è la matrice dei percorsi: riporta i dati di tabella 4.6.

- *S* è la matrice degli stati, inizializzata con tutti zeri e un numero di stati abbastanza alto da poter contenere, secondo previsione, il numero degli step che conterrà una simulazione sul sistema di una settimana (7 giorni). Abbiamo impostato un numero di righe (stati) pari a 10k, - che si è rivelato essere una forte sovrastima quindi va bene -, mentre il numero delle colonne è pari a 43, così organizzate:

- i valori 1-15 rappresentano il fatto che in dato stato il manufacturing task sia in lavorazione e con quale operatore (per i task che richiedono due operatori è indicato il primo, ma l'informazione su quali operatori sono assegnati a ciascun task in ciascuno stato ricordiamo essere contenuta nella variabile *Wt*);
- i valori 16-33 sono invece i transport task, per i quali vale lo stesso discorso dei precedenti;
- i valori 34-41 sono riservati per i task che risultano da assegnare, coerentemente con la variabile *daassegnare* che vedremo nel seguito: sono 8 posizioni riservate, ma vedremo che nelle nostre simulazioni, al massimo 4 posizioni verranno occupate (cioè al massimo 4 task risultano da assegnare in un unico stato);

- il valore 42 indica il numero di job completati, per cui aumenterà con lo scorrere della simulazione fino ad arrivare al valore finale;
- l'ultimo valore, -43-, segna la fine della simulazione e del codice, in modo che esso possa terminare.

La prima riga di questa matrice rappresenta lo stato iniziale del sistema.

Continuando sulle variabili:

- *daassegnare*, - anticipata -, indica, per ogni stato, quali task sarebbero da assegnare: la definizione dei task da assegnare terrà conto dei vincoli della struttura del sistema e dell'occupazione o meno delle stazioni.

- *daassegnareconprio*, contiene gli stessi dati di *daassegnare*, ma ordinati in base alla priorità di assegnazione, che si basa sulla durata residua attesa del percorso al quale il task appartiene: maggiore la durata residua attesa, più alta la priorità, e viceversa.

- *assegnatoconprio*, tiene traccia, per ogni stato, dei task che effettivamente vengono assegnati, anch'essi ordinati per priorità. Capita infatti, che dei task che sono da assegnare in un dato stato, non possano poi effettivamente venire assegnati per mancanza di operatori a sufficienza disponibili. Dunque, *assegnatoconprio* conterrà nelle sue righe gli stessi dati di *daassegnare* e *daassegnareconprio*, ma con delle esclusioni.

- le varibili *durate*… memorizzano le durate residue attese dei percorsi ai quali appartengono i task che risultano da assegnare, in modo da poter essere poi ordinate per priorità e consentire di prioritizzare i task da assegnare. Ad ogni iterazione, queste variabili, che sono dei vettori, vengono riportate alla condizione nulla, in modo che allo step successivo non vi siano memorizzazioni precedenti erronee. Vi è una variabile di tipo '*durate…'* per il prodotto principale ed una per ciascun WIP.

[Tralasciamo del tutto le variabili prettamente tecniche, utili solamente per la scrittura del codice e la sua esecuzione]

- le variabili da *trackcompletamento* a *asswip#*, in numero pari a quello dei jobs che possono essere processati eventualmente tutti insieme nel sistema, a coppie, sono le matrici dinamiche, -con valori come da tabella 4.7-, che seguono l'andamento del singolo job (principale o WIP).

Si decide a priori quanti WIP si vuole ammettere nella produzione: noi in partenza supponiamo di consentire la produzione di 1job+4WIP. Per ogni job consentito, dunque per ciascuno dei 5 sotto questa ipotesi

- *A* è una variabile di tipo cell array che memorizza per ciascuno stato le azioni che è possibile intraprendere in tale stato;

- le azioni sono memorizzate, in modo univoco, nella matrice *azioni*, man mano che vengono scoperte;

- *t* è la variabile che tiene traccia del tempo simulato, dunque cresce fino al tempo in cui viene interrotta la simulazione. Gli step temporali saranno pari al numero di stati;

- *ttemp* memorizza l'avanzamento temporale per ciascuno stato;

- *pd* (che sta per *probability distribution*) è una variabile di tipo cell array costruita in modo da contenere le distribuzioni di Simpson di probabilità dei tempi lavorazione dei 33 task;
- *tempi* è una variabile di tipo matrice con corrispondenza alle prime 33 colonne di quella *S*, in cui per i tali posizioni è indicato il disavanzamento temporale dei task in questione (quando il task sarà completato il valore tornerà ad essere nullo in entrambe le matrici) in lavorazione in determinati stati;
- *i* è importante da specificare, pur essendo una variabile prettamente tecnica, perché indicherà lo stato.

4.2.2 Il codice

Adesso spieghiamo il codice, - non in tutti i dettagli perché sarebbe troppo lungo -, nel suo funzionamento e nelle sue parti.

L'intero codice è racchiuso in un ciclo while con condizione sulla variabile booleana END, impostata a zero e che assumerà il valore 1 quando la condizione di terminazione (che vedremo alla fine) sarà soddisfatta.

Una volta entrati nel ciclo, la prima cosa che il codice fa è verificare quali task siano in corso e memorizzare il risultato nella variabile di tipo cell array incorso. Questa memorizzazione è utile, usata insieme ad altre, - come *S*, *finisce* e *assegnatoconprio* -, per fare verifiche sul codice o su particolari suoi punti.

La prima parte del codice, - righe 98-670 -, - al di là del settaggio delle variabili ausiliarie a 0 o a 1- , è costituita da tanti cicli while quanti sono i job autorizzati nel sistema, quindi in questo caso abbiamo 5 cicli while: il primo è sempre relativo al job principale, e i successivi sono, in ordine, relativi ai 4 WIP.

Ciascun ciclo while, - quindi ciascuno dei 5 eventuali job-, ha un proprio indice, in ordine: *k*, *q*, *qq, q3, q4,* che segna l'avanzamento del job (quindi parte sempre da 1) secondo la colonna della relativa matrice che traccia il completamento (in ordine: *trackcompletamento, trackwip, trackwip2, …, trackwip4*): ad ogni step il codice verifica se nella colonna attuale (identificata come abbiamo detto da *k*, o da *q*, o da *qq,* o da *q3,* o da *q4*) della relativa matrice di tracciamento del completamento del job vi siano ancora degli 1, ossia dei job (magari anche assegnati ma) non ancora completati, e in caso contrario incrementa la variabile di 1 ad indicare che è da quella colonna in poi che è possibile trovare dei task non ancora completati.

Lo scopo di questi cicli while è la ricerca, -effettuata per ciascun potenziale job-, dei task che sarebbero da effettuare; essa include tutte le verifiche sul sistema che consentono di determinare che un determinato task che risulta da assegnare dalla matrice che traccia l'assegnamento, sia effettivamente potenzialmente assegnabile, ossia i suoi task predecessori siano già stati completati, e la stazione su cui deve essere effettuata la lavorazione, -o a cui deve arrivare il trasporto, in caso si tratti di un task di trasporto- , non sia occupata (o verrebbe occupata da un task in competizione per la stessa risorsa, relativo ad un job prioritario, quindi già determinato da un ciclo while precedente).

Ciascun ciclo while dunque, ha una condizione di verifica costituita da una variabile, - *Q*, la stessa per tutti perché viene ad ogni ciclo controllata e riportata al valore nullo- , e dall'indice *k*, o *q*, o *qq,* o *q3,* o *q4* di cui prima, che deve essere minore di 14, perché 13 è l'ultima colonna della tabella dei percorsi relativa ai task.

Ciascun ciclo quindi va avanti a controllare per la presenza di task da assegnare, sino a che l'indice è al massimo pari a 13, e contemporaneamente *Q* non sia diventato 1. Come vediamo quasi al fondo di ciascun ciclo while, la variabile di controllo *Q* diventa 1, interrompendo il proseguo dell'attuale ricerca, quando la colonna della matrice di tracciamento del completamento (esclusa la colonna 1) in esame è stata esaminata tutta, non è stato trovato un task da assegnare e il numero di elementi non nulli della medesima colonna della matrice che traccia l'assegnamento del job in considerazione è pari al numero di elementi non nulli della medesima colonna della matrice dei percorsi, che vuol dire che è inutile proseguire nella ricerca di task da assegnare nelle colonne successive perché non si troveranno e questo tipo di ricerca nel complesso risulterebbe inutilmente onerosa. Altra condizione che fa sì che *Q* passi a 1 e che quindi la ricerca termini è quella naturale, ossia che le colonne siano state analizzate tutte e quindi l'indice si trovi (momentaneamente, infatti al termine del ciclo viene riportato al suo valore di partenza, in modo che non si perda l'informazione sulla colonna della matrice che traccia il completamento da cui partire nella ricerca) al valore 14.

Per quanto riguarda la ricerca dei task da assegnare, viene analizzata la matrice che tiene traccia del completamento partendo dalla colonna indicata dall'indice, come spiegato appena; per i task non ancora completati e non ancora assegnati (si guarda alla corrispondenza nella matrice che tiene traccia invece dell'assegnamento), e che risultino completati per il job precedente (questa condizione vale ovviamente solo per i job di tipo WIP, in quanto il job principale è il primo in priorità) (si guarda sempre alle matrici del completamento) si va a verificare che sia effettivamente possibile svolgerli, tenendo conto dei vincoli di sistema. Il significato di questa ricerca è:

- se il task in questione è un *manufacturing task*, si verifica che la stazione su cui deve essere lavorato sia libera, fatta eccezione per i *mt* di saldatura 8 e 11, per i quali la stazione risulta necessariamente già occupata perché sono preceduti dai task di saldatura 9 e 10, rispettivamente, e per i task di consolidamento e assemblaggio 14 e 15, anch'essi preceduti da task che devono essere eseguiti sulla medesima stazione, ovvero i task 12 e 13, rispettivamente;
- se il task in questione è un *transport task*, si verifica che sia uno dei due finali (32 e 33), oppure non finale ma per il quale la stazione di arrivo è libera (variabile *St*) e allo stesso tempo non risulta già da assegnare un task che la occuperebbe prioritariamente (variabile *Stt*) (quindi questo discorso vale solo per i WIP in quanto il job principale ha priorità su tutto), oppure si tratta del *tt* 18, che da solo costituisce un percorso, e il task suo task predecessore (il task 10) è già stato completato (quindi può essere assegnato).

Per i task che rientrano nel secondo punto, se si tratta del task 25 o del 27, che portano ai due processi di saldatura, quindi hanno una funzione analoga ma in due percorsi differenti, se uno dei due risulta già da assegnare, o se si tratta del task 27 e:

- non vi è un percorso di lavorazione assegnato per il job in questione (ossia la variabile di tipo *perc*… relativa al job in questione è nulla)
- il percorso di lavorazione 1 (*w11*) non è nullo, -quindi vi è un job in corso sul percorso 1-, o la stazione 11 (quella su cui si svolge tal percorso di lavorazione) non è libera

- il percorso di lavorazione 2 (*w12*) non è nullo, -quindi vi è un job in corso sul percorso 2-, o la stazione 12 (quella su cui si svolge tal percorso di lavorazione) non è libera

e per tutti i task che non soddisfano nessuna delle due condizioni, il ciclo termina e il task non verrà assegnato. La parte successiva del ciclo while per questo tipo di task non viene eseguita: la variabile di controllo *go* viene posta a 0.

Per i task per cui il controllo può proseguire, dunque in quei casi in cui la variabile di controllo *go* viene posta dal codice pari a 1 si prosegue con un altro controllo preliminare: per i *tt*, fatta eccezione di quelli che hanno come stazione di arrivo una delle due stazioni dove si svolge il consolidamento e l'assemblaggio, quindi quei task che si trovano all'inizio dei percorsi di lavorazione alternativi nella parte finale del processo, ossia i task 20, 21, 28, 29, 30 e 31, si verifica che il task da assegnare in analisi non abbia come stazione di arrivo la stessa stazione in cui deve arrivare un task di trasporto che risulta già in corso (si fa uno scan sulla variabile *Wt*), nel qual caso, go diviene 0 in quanto la stazione nella pratica non è disponibile. Questa situazione di 'prenotazione' della stazione di arrivo da un *tt* già assegnato ma non ancora terminato non è un problema per i task 20, 21, 28, 29, 30 e 31 perché il consolidamento e assemblaggio richiede che l'output dei 3 rami 1, 3 e 6 o 2,4 e 7 giunga complessivamente nella stazione 11 o 12 rispettivamente per essere appunto assemblato e consolidato e terminare così il proprio processo.

Arriviamo adesso al controllo vero e proprio, riservato ai task per i quali non vi sono problemi a livello di sistema per l'assegnazione, per i quali la variabile di controllo *go* è stata posta e lasciata pari a 1.

Per questi task si va ad effettuare la verifica sui predecessori, in modo da determinare se siano assegnabili, nel rispetto dei vincoli di precedenza del processo. Si pone dunque dentro *pred* l''elenco' dei predecessori del task in questione, attingendo dalla matrice dei task che, - tra le altre -, contiene questa info.

- Se il task in analisi non ha predecessori (solo i task della colonna 1 della matrice dei percorsi *P* soddisfano questa condizione) allora esso può sicuramente essere assegnato;

- in caso contrario si effettua uno scan di tutti i predecessori e: innanzitutto si verifica che sia stato completato; in caso affermativo, si incrementa di 1 la variabile di controllo *ok* che serve a verificare se tutti i predecessori del task in analisi sono stati completati, come richiesto dalle precedenze; poi, si verifica se il task in questione è il numero 3, il 9 o l'11 (la quale indicazione è data dalla presenza del numero 2 in colonna 8 della matrice *task*), dai quali partono i task di trasporto che portano alle stazioni di consolidamento e assemblaggio, per i quali bisogna fare un distinguo in quanto, -come abbiamo spiegato anche in precedenza, ed è un punto importante-, una volta scelto uno dei due percorsi di lavorazione per la parte finale, non si vadano ad assegnare task relativi ad un altro percorso di lavorazione finale. Per evitare questo problema, se il task sotto valutazione appartiene al primo percorso di lavorazione finale (che viene assegnato a meno che non lo sia già stato, quindi ha priorità sul secondo in generale), - si tratta quindi dei task 31, 21 e 29-, e il *mt* di consolidamento e assemblaggio che viene dopo (quindi il 12 in questo caso) non è in corso (lo si vede da *S*) si procede a controllare che non risulti già da assegnare almeno 1 dei task del percorso di lavorazione gemello (quindi i task 30, 20 e

28), e in caso positivo si pone a 0 la variabile *ok2*, il che farà sì che non si proceda nell'assegnazione del task in esame.

Situazione analoga vale se i task in questione sono quelli che portano alla stazione 12, cioè i *tt* 20, 28 e 30, con la sola differenza che ad essere controllati saranno i task relativi al percorso gemello a questo (cioè quello in questione prima).

Questo pezzo di codice modella il fatto che non si possono potenzialmente scegliere due percorsi gemelli per lo stesso job, ed evita che risultino da assegnare (e quindi eventualmente poi vengano assegnati) due o più job tra i job 20, 21, 30, 31, 28 e 29 appartenenti a entrambi i percorsi di lavorazione in coda.

A questo punto, se il valore della variabile *ok* è pari al numero dei predecessori (ossia per tutti quanti si è avuto l'ok) e ok2 è pari a 1 (che significa che *primo* è andato a 0 in caso di analisi dei 6 task di trasporto 'speciali') si fa l'ultima verifica su task, che è quella sul task 18, che ricopre un ruolo particolare, e si guarda all'avvenuto completamento del suo predecessore, ovvero il task 10 e in caso positivo si procede con l'assegnamento.

L'assegnamento prevede l'inserimento del task nella variabile *daassegnare* e dell'informazione sulla durata del percorso di cui fa parte nell'apposita variabile di tipo 'durata', entrambi nella posizione *z*, l'incremento dell'indice *z* di 1 in modo che l'eventuale successivo task che risulterà da assegnare in quello stato e la relativa info sulla durata andranno ad essere inserite nella posizione immediatamente successiva, e il porre la variabile di controllo *trovato* pari a 1, ad indicare che è stato identificato (almeno) un task da assegnare. Per i *manufacturing task* (esclusi i task 8 e 11), e per i *tt* 25 e 27, viene anche indicato (modifica della variabile *Stt*) che vi sono da assegnare task che richiederebbero la relativa stazione, e l'indicazione del numero (1 per il job principale, 2 per il primo wip e così via) dà anche la info su quale job ha riservato la stazione. Due job che competono per la stessa stazione ma sono relativi al medesimo job, effettivamente competeranno, mentre se si tratta di due job diversi che competono per la stessa stazione, il problema non si andrà a porre in quanto il meno prioritario non verrà direttamente assegnato.

Se il ciclo while non ha identificato alcun task da assegnare, allora la variabile di tipo '*stepup*' relativa al job in questione diventa pari a 1. Per chiarezza: se ad esempio il secondo ciclo while non ha identificato nessun job da assegnare, la variabile *stepup2* viene posta pari a 1 (le variabili di tipo *stepup* sono inizializzate a 0 ad ogni macrociclo, ossia per ogni stato esplorato).

Alla fine di ogni ciclo while viene calcolato e memorizzato il numero di task da assegnare per quel ciclo; le variabili a questo scopo sono: *b1, b2, …, b5*, e mantengono anch'esse l'ordine di priorità dei job: ad esempio, la variabile b3 ci dirà quanti task sono da assegnare per il WIP numero 2 nello stato i (ma non teniamo traccia dei valori per ogni stato, perché non ci serve; queste sono variabili locali).

Facciamo notare infine tra i (5 in questo caso del modello di partenza) cicli while della parte iniziale di cui abbiamo parlato è possibile trovare qualche piccola differenza nei codici (oltre ovviamente a differenze in indici e variabili necessarie a differenziare le situazioni dei job) dovuta al fatto che per il

primo e l'ultimo job qualche calcolo è inutile, e che il fatto che si tratti del job principale piuttosto che di un WIP, o che si tratti del primo WIP in ordine piuttosto che del quarto, richiedono modellizzazioni lievemente differenti.

La seconda parte del codice, - righe 672-1279-, si occupa, -a grandissime linee, che diano subito un'idea generale di questa parte-, dell'assegnazione dei task agli operatori e della determinazione delle azioni per lo stato in questione.

Questa seconda parte incomincia con il check di una condizione, che è: `(stepup1~=1) || (stepup2~=1) || (stepup3~=1) || (stepup4~=1) || (stepup5~=1)` (1). Il fatto che questa condizione sia verificata significa che vi è almeno un task da assegnare risultante dalla parte 1. In caso non sia verificata questa condizione, il codice va direttamente alla riga 1264, ed esegue un numero limitatissimo di operazioni, in quanto non essendoci task da assegnare, -a causa dei vincoli di precedenza o di altre cause da ricercarsi nella parte 1-, non è possibile eseguire alcuna azione (o meglio, l'azione eseguita sarà quella di assegnare operatori a ulteriori task) e il sistema evolverà nel successivo stato naturale. Le poche operazioni che Matlab deve effettuare, - e sono comuni anche al caso in cui la condizione (1) sia verificata, ossia vanno eseguite in ogni caso, e vanno specificate due volte per ciò-, sono espresse nelle righe 1265-1271, appena dopo la parola chiave `else`, e nelle righe 1157-1163. Queste espressioni trasportano dati dello stato corrente *i* a quello successivo *i+1*, e precisamente i dati relativi a: l'occupazione o meno degli operatori (variabile *W*), il task con cui un operatore è eventualmente occupato, lo è (*Wt*), la posizione corrente degli operatori (*Wp*), l'occupazione dei task (variabile *S*, posizioni 1-33), il numero di job attualmente completati (matrice *S*, posizione 42), l'indicazione per i task in corso dell'appartenenza a quale job (variabile *s*), e l'indicazione delle stazioni attualmente occupate (variabile *St*). Facciamo notare che queste variabili fondamentali non rimarranno così sino al successivo stato, ma verranno ulteriormente modificate nella parte 3 del codice, in conseguenza a quale task termina in questo stato (c'è sempre un task che termina in un determinato stato).

Ammettendo invece che la condizione (1) sia verificata, si procede innanzitutto a copiare i valori di *daassegnare* (i,:) nelle apposite posizione della matrice *S*, relativamente a quello stato; poi, come anticipato, si ordinano i valori delle durate dei task da assegnare in maniera da costruire il vettore dei task da assegnare ma con priorità, *daassegnareconprio*, e appunto lo si costruisce, copiando i valori di *daassegnare* (i,:), in ordine di priorità di job, nelle medesime posizioni del vettore *daassegnareconprio*(i,:), ma ordinati secondo priorità di assegnazione. *daassegnareconprio* ha un numero di colonne doppio rispetto a *daassegnare;* questo perché la seconda metà è designata ad ospitare un valore (in corrispondenza di ciascun task da assegnare che troviamo nella prima metà, quindi si mantiene l'ordine di priorità di assegnazione) che indica il numero di operatori da assegnare. Il numero di operatori da assegnare può essere solamente 1 o 2 per gli operatori standard, mentre il fatto che si tratti del saldatore viene indicato dall'apposizione del valore '3'.

Successivamente, inizia la parte che serve per assegnare, -secondo disponibilità-, gli operatori ai task che risultano da assegnare: questa parte è compresa nelle righe di codice 716-898. Si tratta di un ciclo while

sulle posizioni di *daassegnare*, che va avanti sino al primo zero che si trova, ossia sinché vi sono task per cui valutare l'assegnazione di uno o più operatori. All'interno del ciclo vi è la suddivisione fra *manufacturing tasks* e *transport tasks*, per i quali le istruzioni saranno in parte uguali, ma non completamente.

- Se il task di *daassegnareconprio* in questione è un *mt*, si fa un'ulteriore suddivisione: i task di saldatura, - quindi i task 8-11 -, sono trattati separatamente preventivamente, perché per essi si deve valutare l'assegnazione del saldatore; allora si verifica se il saldatore (posizione 10 del vettore *W*(i,:)) sia libero (quindi vi sia uno zero in corrispondenza di tale posizione) e non precedentemente assegnato nello stato corrente (variabile *assegnato*, check posizione 10). In caso affermativo della verifica, si può procedere ad assegnare il saldatore al task in analisi, e dunque:

  - si inserisce il suo codice (10) nella riga corrispondente allo stato corrente della matrice *assegnprioritaria*, che memorizza per ogni stato l'assegnazione prioritaria degli operatori;

  - si segnala che l'operatore 10 è stato assegnato (quindi non sarà più assegnabile nel seguito del ciclo) (variabile *assegnato*);

  - analogamente, si occupano le decime posizioni delle matrici *assegnatoop* e *assegnatotask*, riga i-esima, rispettivamente con il valore '10' e con il numero del task che si sta assegnando;

  - punto importante: si inserisce il task che è stato possibile assegnare in *assegnatoconprio* (i,:);

  - si compila il valore in *s* che corrisponde al tipo di job a cui è relativo il task che stiamo assegnando (principale, WIP, ecc.), deducendolo dalla matrice da, preventivamente compilata nella riga i-esima con il valore da passare.

  Per gli altri *manufacturing task* (quindi quelli non di saldatura) si va ad assegnare l'operatore/i più vicino/i o che percorrerà il minor numero di chilometri tra quelli disponibili. Per far ciò si calcola la distanza che si troverebbe a percorrere ciascun operatore potenzialmente assegnabile, e si assegna quello per cui essa risulta più piccola possibile. Questa parte include varie differenziazioni:

  - se il task da assegnare richiede un solo operatore, l'operatore per cui la distanza risulta minima può essere subito assegnato;

  - se il task da assegnare richiede due operatori e abbiamo fatto la valutazione sul primo trovando un risultato, memorizzo il numero del primo operatore che andrei ad assegnare nella variabile temporanea *val_1*, e lo segno come assegnato;

  - se il task da assegnare richiede due operatori e abbiamo valutato anche il secondo trovando anche in questo caso un risultato, vado ad effettuare l'assegnazione;

  - se invece il task da assegnare richiede due operatori ma non è stato possibile trovare il secondo operatore da assegnare, tolgo il blocco all'operatore che avevo ritenuto assegnato quando non avevo ancora effettuato la ricerca per il secondo, e lo rimuovo anche dalla variabile *assegnprioritaria* in cui l'avevo inserito, in quanto nella pratica non verrà assegnato, dato che manca il compagno/la compagna per lo svolgimento.

  L'assegnazione in questi casi è la medesima di quella spiegata per i task di saldatura, con differenze dovute solo agli indici di assegnazione.

- Se il task di *daassegnareconprio* in questione è un *tt*, la procedura è la medesima rispetto alla precedente, esclusa la parte speciale relativa alla saldatura.

Ad ogni fine ciclo viene aggiornato il valore della variabile *p*, che parte da zero ad ogni nuovo stato (riga 717), e serve per posizionare in modo corretto i secondi operatori, -per quei task che ne richiedono due -, nella matrice *assegnprioritaria*; viene ovviamente incrementato di 1 il valore dell'indice sul quale si svolge il ciclo (*j*), e si riportano a zero i valori delle due variabili temporanee *val* e *val_1*.
Questo è il termine della procedura di assegnazione degli operatori ai task.

Adesso, dopo aver trascritto la riga in questione di *assegnatoconprio* in modo che non risultino zeri tra un task assegnato e l'altro, il che darebbe problemi nel successivo ciclo, si procede, per ciascun task che è stato assegnato (righe 937-1094), se non è ancora stato fatto in precedenza (check sulle variabili di tipo *perc*…), se si tratta di uno di quei 6 task di trasporto che portano al consolidamento e assemblaggio e implicano la scelta di un percorso e l'automatica esclusione del percorso gemello, ad annullare tutte quelle posizioni delle matrici che tracciano il completamento e l'assegnamento relative al percorso gemello di quello che sta essendo scelto, i cui task non dovranno essere eseguiti.
L'assegnazione del percorso è prioritaria sul percorso 12 (task di trasporto che vi arrivano: 21,29,31) e sul job (il job in lavorazione viene valutato prima di tutti, il primo WIP dopo e così via).

Conclusa questa parte, - fondamentale per il codice -, si passa alla parte relativa alle azioni, compresa nelle righe 1095-1153. Questa parte ha l'utilità di determinare le azioni che è possibile intraprendere in ciascuno stato, seppur l'azione sia stata già scelta dall'algoritmo nella parte di assegnazione, e la scelta non verrà modificata. Questo calcolo è utile in vista del Reinforcement Learning, e di una valutazione di assegnazione diversa degli operatori da quella derivante dalla scelta della politica allocativa della *Shortest Distance*.
Innanzitutto, si trova il numero di operatori che risultano assegnati (variabile *KK*(i)), contando il numero di elementi non nulli di *assegnatotask* (i,:) (si usa la funzione matlab *nnz*, che sta per '*number of non zeros'*), ma escludendo dal conteggio quelli di saldatura, per i quali non è necessario scegliere alcuna azione, in quanto se vi è un task assegnato, viene assegnato il saldatore sicuramente.
- Se *KK*(i) risulta diverso da zero, ossia vi è almeno un task non di saldatura assegnato:

- Innanzitutto si trova il numero di operatori (standard) disponibili nello stato corrente (variabile *nopdisp*(i)), sottraendo al totale ideale degli operatori standard, - quindi 9 -, il numero di quelli attualmente impegnati (*nnz*(*W*(i,1:9))); poi si crea un vettore *v* costituito dagli operatori disponibili, ordinati, e lo si copia nella riga i-esima della matrice *vi*, che si occupa di memorizzare per ogni stato appunto quali operatori sono disponibili. Se il saldatore non è occupato, viene aggiunto in coda anche l'operatore 10 (il suo numero).
- A questo punto si determina, attraverso la funzione *npermutek,* che calcola, per gli input che le diamo, le disposizioni senza ripetizione degli operatori *v* in *KK*(i) posizioni, che andranno a determinare la matrice *NPK*, che verrà sovrascritta ad ogni *i*.

Se il saldatore risulta tra gli operatori da assegnare, si crea una colonna *col* costituita da tante righe quante sono le righe di *NPK* (valore precedentemente inserito nella variabile *r*), e si incrementa di 1 la variabile *c* che invece indica il numero di colonne di *NPK*.

NPK dunque, è una matrice che contiene tutti i possibili modi in cui è possibile assegnare gli operatori attualmente disponibili ai task da assegnare.

- Adesso, si va a costruire una matrice simile a *NPK*, nella quale viene inserita la colonna *col* creata precedentemente, in corrispondenza della posizione in cui l'assegnazione è 'operatore 10' ossia il saldatore, in modo da riflettere perfettamente la scelta delle azioni. Questo nel caso in cui vi sia tra gli operatori assegnati anche il saldatore, altrimenti vengono copiate una ad una le colonne di *NPK*.
- Si memorizza *NPKn* (la matrice output del punto precedente) nella posizione i-esima della variabile *A* delle azioni.
- Si completa *NPKn* con tante colonne di zeri fino a renderla una matrice di 8 colonne come quella delle azioni (*azioni*), il che è necessario perché possano essere confrontate automaticamente. Si guarda dunque a quali elementi (ragionando sulle righe) di *NPKn* siano membri di *azioni*, utilizzando la funzione *ismember* con opzione sulle righe ('rows'), e i risultati vengono salvati nella variabile *I*, che conterrà un 1 nelle posizioni di riga per cui la riga risulta membro e uno 0 in quelle per cui invece non risulta membro. *I* viene poi trasformata in una variabile che contiene gli indici di riga che a cui corrispondono i propri elementi non nulli, - quindi gli indici di riga di tutti gli 1, in ordine-, usando la funzione *find*, e le righe corrispondenti ai valori di *I* vengono rimosse da *NPKn,* che a questo punto conterrà solo righe (cioè azioni) non preventivamente inserite nella matrice *azioni.*

  Viene aggiornato il numero di righe di *NPKn*, aggiornando *r* sottraendole il numero di elementi di *I*.

  Infine, viene aggiornata la matrice *azioni* con le nuove azioni e aggiornato l'indice *in* che indica il punto da cui partire la volta successiva a inserire le azioni in *azioni*, sommandogli il valore *r*.

- Se invece *KK*(i) risulta pari a zero, il che significa che vi è solamente un task di saldatura da assegnare e nient'altro, si procede ad indicare '10' in *A*, in quanto l'unica azione che è possibile intraprendere in questo caso è assegnare il saldatore, si determina il *vi*(i) dello stato che è dato dal vettore [10,0,0,0,0,0,0,0,0,0], e se non è già presente in *azioni*, lo vi si inserisce.

Terminata la parte relativa alle azioni, si calcola il numero di operatori assegnati e si procede con le stesse assegnazioni allo stato successivo che abbiamo visto all'inizio della parte 2, quando abbiamo parlato del check sulle variabili *stepup*… (righe 1154-1164).

Alla riga 1165 inizia un ciclo *for* sugli operatori assegnati e i relativi task in cui:

- Si segna nello stato successivo *i+1* che il task in questione è stato assegnato all'operatore in questione (matrice *S*);
- se si tratta di un *manufacturing task*:

  - se si tratta del *manufacturing task 12* o *13*, viene liberato il percorso relativo, -w12 o w1, rispettivamente-, in quanto tutti e 3 i *tt* che vi arrivano sono sicuramente stati completati, e allo stesso modo viene eliminata l'informazione sul job a cui si riferiva la specifica del percorso;

- viene impegnata la stazione di lavoro del task in questione;

- se si tratta di un *transport task* (ma non il 17, per il quale la stazione di partenza non va liberata in quanto può essere liberata solo dopo l'esecuzione del task 18, che per come è strutturato il processo avviene sicuramente successivamente):
  - si libera la stazione di partenza del *transport task*; per i task 25 e 27 viene inserita un'eccezione in modo tale che non possa venir assegnato il task di saldatura successivo, ossia il 9 o il 10 sinché non termina quello di trasporto (righe 1200-1216 e 1347-1354);
  - se si tratta di uno dei task 31, 21 e 29, o di uno dei task 30, 20 e 28, si pone a uno la variabile di controllo *w11*, -o *w11* rispettivamente-, e si ricerca e poi segnala nella variabile di tipo *perc…* a quale job è relativa la segnalazione del percorso;
  - infine, si identificano gli indici di percorso del task assegnato in considerazione e, a seconda del job a cui è relativa l'assegnazione, si procede ad annullare la relativa casella della opportuna matrice che traccia l'assegnamento, in modo da segnalare che tale task è già stato assegnato e non risulti più da assegnare.

L'ultima operazione relativa a questa parte 2 è (righe 1259-1266), per ciascun task assegnato, simulare un tempo di processo dalla relativa distribuzione, -ed inserirli in ordine nell'apposita variabile rand-, e inserire poi quello stesso valore nella matrice dei tempi, in corrispondenza della riga i e del task in questione. Si verifica anche se si tratta della prima volta in cui si simula il tempo di processamento di un certo task (attraverso check dell'apposita variabile *rngen*, inizialmente appositamente impostata con tutti i valori a 1), nel qual caso il *random number generator* di Matlab viene impostato su '*default*' (comando: `rng('default');`) e poi si va a segnalare che la prima simulazione è già avvenuta, per il controllo successivo.

L'ultima parte del codice identifica quale job termina, effettua le necessarie modifiche alle variabili, ed incrementa lo stato. Precisamente:

- si inseriscono nella variabile *randoms* i valori dei tempi residui di processamento dei task in corso, e i relativi indici di colonna nella variabile *ind*, applicando la funzione *find* alla riga i-esima della matrice dei tempi *tempi;* i tempi vengono poi ordinati e memorizzati nella variabile *randorder* (`randorder=floor(tiedrank(randoms));`); si memorizza in *J* il valore minimo di *randorder* e si memorizza l'avanzamento temporale dello stato corrente, aggiornando la riga i-esima della variabile appositamente creata e introdotta nella sezione dedicata all'esplicazione delle variabili, *ttemp*. I tempi dei task assegnati nello stato corrente, che nella riga 1265 abbiamo assegnato allo stato successivo, secondo la variabile *rand*, vengono tutti aggiornati sottraendogli il valore corrente di *ttemp*.

- Adesso vengono effettuate tutte le operazioni dovute al terminare di uno dei task in corso:

  - per ciascun task che finisce (potrebbero infatti finire insieme due task o più, e anche se questa eventualità è rara va tenuta in considerazione) viene segnalata la sua fine nello stato corrente nell'apposita matrice *finisce*;

- si identifica la sua posizione nella matrice *P* dei percorsi e grazie ad essa, dopo aver identificato il job della cui matrice di track completamento segnalare la conclusione del job:
  - si annulla la corrispondente posizione della matrice che traccia il completamento;
  - se il job non è quello principale (per cui il valore identificativo nella matrice *s* è 1) si riporta al valore standard (1) la posizione in *s* che era stata modificata per il job in questione;
  - si annulla la posizione relativa al task nella matrice *S.*
- Si identifica e memorizza nella variabile *f* l'indice/gli indici della/e posizione/i del task che finisce in questione in *Wt*(i+1,:), e per esse, se si tratta di un task di trasporto, non di manifattura, si modifica la posizione di arrivo, -quindi la variabile *Wp* nell'istante successivo *i+1*-, con la stazione di arrivo del *tt* che termina, e si annullano le relative posizioni nell'istante successivo *i+1* delle variabili *W* e *Wt*.
- Usando sempre la funzione *find* per l'identificazione degli indici, -ma della matrice *daassegnare* in questo caso-, si trovano gli indici di riga e colonna dell'ultimo task del tipo in analisi assegnato e, sempre identificato anche a quale job ci si riferisce,- mediante la matrice *s*-, si procede a diminuire la durata del relativo percorso nella matrice che traccia il completamento per il job in questione, facendo attenzione a verificare che in caso il valore diventi negativo, esso venga riportato positivo, in quanto in caso contrario si comprometterebbe la procedura per le successive assegnazioni di task per priorità di durate.

- Si conclude la procedura iniziata per i *tt* 25 e 27: si controlla che la variabile *Stt* nell'istante corrente e per la stazione 6 sia impegnato e che la variabile di controllo *pass* sia diversa da 0 e, se *pass* è presente tra i task che finiscono, la variabile di controllo viene riportata a zero, altrimenti il valore di *Stt* corrente viene riportato all'istante successivo. Zeta viene in ogni caso rimessa a 0.

- Si fa uno scan sulle colonne delle matrici che tracciano il completamento di ciascun job a cui si è giunti, -quindi identificate per ciascun job rispettivamente dagli indici *k, q, qq, q3 e q4*-, per determinare se i valori siano tutti nulli (quindi tutti i job della colonna siano stati completati e si può passare alla successiva), nel qual caso l'indice apposito viene incrementato di 1.

A questo punto si arriva alla parte finale vera e propria, in cui l'indice di stato viene incrementato di 1 (riga 1393), per cui è come se si fosse già nello stato successivo: si effettuano delle operazioni preventive alla ricerca dei task nello stato successivo, e si verifica che non si sia giunti allo stato finale, nel qual caso il codice si arresta e il marco ciclo while sullo stato successivo non viene eseguito.

Le operazioni che si eseguono in questa parte (1394-1469) sono:

- viene memorizzato il tempo dello stato successivo (*i+1*) che sarà dato dal tempo dello stato precedente *i* più il valore di *ttemp* dello stato precedente *i*;

- viene aggiornata la riga del nuovo stato corrente *i+1* della matrice dei tempi *tempi*, sottraendo ai valori dello stato precedente *i* sempre il valore di *ttemp* dello stato precedente;

- si controlla la matrice del tracciamento del completamento del job principale per verificare se abbia finito, quindi si verifica se la 13-esima colonna di *trackcompletamento* sia completamente nulla (inteso come "costituita da tutti zeri"), nel qual caso, -poiché significa che il job è stato terminato-, si procede a:

- incrementare di 1 unità la variabile *produzione*;
- inserire nella variabile *out* il tempo corrente, che è quello in cui viene terminata la produzione del job numero 'produzione';
- si aggiorna la posizione numero 42 della matrice *S* degli stati, che è appunto quella dedicata all'indicazione della produzione, con la nuova produzione (variabile *produzione*);
- si aggiornano tutte le matrici di tracciamento di assegnamento e completamento dei job: poiché il job 1 si considera uscito dal sistema, il job2 diventa il primo, il job3 il secondo e così via, e un nuovo WIP potrebbe da questo punto in poi essere lavorato. Si copiano dunque i dati di *trackwip* e *asswip* in *trackcompletamento* e *trackassegnamento*, quelli di *trackwip2* e *asswip2* in *trackwip* e *asswip*, …., e *trackwip4* e *asswip4*, cioè le matrici di tracciamento di assegnamento e completamento del potenziale nuovo WIP vengono inizializzate allo stesso identico modo in cui sono state inizializzate nella definizione delle variabili (righe 69-70); analogamente si modificano gli indici: *q* diventa *k*, *qq* diventa *q*, *q3* diventa *qq*, *q4* diventa *q3* e *q4* diventa1. Tutto questo rientra nelle righe 1400-1416.
- anche la matrice *s* va aggiornata, modificando i 2 in 1, i 3 in 2 e così via, per cui si fa uno scan su tutte le colonne della sua riga corrente (cioè la i-esima) e si modificano i tal senso tutti i valori diversi da 1;
- lo stesso discorso vale con i percorsi: si va a verificare per i job dal 2 in poi (cioè i WIP), se vi sia uno dei due percorsi di lavorazione in coda al sistema assegnati, e in tal caso lo si annulla e assegna alla variabile di tipo *perc* relativa al job precedente.

Alla riga 1463 vi è il controllo che abbiamo impostato su questo codice per il suo termine, che è quello di avere una produzione pari a 145. Dunque, se la produzione è arrivata a 145, la variabile di controllo *END* viene posta pari a 1 e il codice termina, in quanto il macro ciclo while, che ha appunto, - come detto all'inizio-, il controllo sulla variabile *END*, non viene più eseguito per cui il codice terminerà la sua esecuzione.

La motivazione per l'aver posto come condizione finale quella sulla produzione pari a 145 è dovuta al fatto di voler avere un margine in eccesso sui dati che ci serve di reperire dalla simulazione, e con questa impostazione siamo sicuri di averli. La nostra simulazione è rivolta a verificare la performance del sistema, con le impostazioni sul numero di job e WIP e sul numero di operatori inserite, in un periodo di 7 giorni lavorativi, con il sistema che parte da fermo; siamo sicuri di ottenere i risultati che ci servono impostando la condizione precedente sulla produzione, in quanto abbiamo osservato per tentativi grossolani che la produzione che è possibile raggiungere sotto tali ipotesi è piuttosto inferiore ad un output totale di 145 unità. Anche i risultati dell'altro studio fatto in precedenza (autonomo rispetto a questo) sul sistema (codice 3. della bibliografia) riportano un output settimanale inferiore a 145, dando ulteriore prova del fatto che la scelta è opportuna.

Volutamente sono state date, appena, informazioni molto generiche sulla simulazione, in quanto molte altre informazioni generiche su di essa sono state date nell'arrivare sino a qui, e perché vi sarà nel seguito un capitolo dedicato interamente alle simulazioni e all'analisi dei relativi risultati ottenuti.

Una volta giunti alla fine del codice, si segnala che si è all'ultimo stato ponendo a 1 la posizione apposita (43) della matrice *S*, riga corrente.

Il calcolo della matrice *xx* che troviamo nell'ultima riga, che trova le righe univoche di S, serve solo per controllare se ci siano stati ripetuti, ma non ha una reale utilità per le simulazioni.

Capitolo 5: Studio del numero di operatori mediante simulazione

Questo capitolo è dedicato alle simulazioni.

Ciò che ci interessa determinare è il comportamento del sistema produttivo al variare del numero di operatori e dei job consentiti all'interno del processo, in modo tale da poter dare un consiglio al titolare dell'azienda sulla necessità o meno di cambiamenti su tali parametri, e se sì di suggerire la modifica più opportuna.

Come detto anche nel capitolo 3, la eventuale modifica del numero di operatori interessa solamente gli operatori standard e non il saldatore, in quanto non vi è possibilità di effettuare modifiche sul layout dello stabilimento, motivo per cui la stazione di saldatura rimane unica ed aggiungere uno o più saldatori per velocizzare il processo (ridurre il *cycle time CT*) non porterebbe ad alcun risultato.

Lo stabilimento produttivo è attivo 5 giorni a settimana, -dal lunedì al venerdì-, e segue gli orari 8.30-12.30 e 13.30-17.30, per un totale di 8 ore. Poiché ragioniamo in termini di secondi nel codice, riportiamo l'espressione in secondi di una giornata lavorativa di 8 ore:

8 h = 8 * 60 min = 8 * 60 * 60 s = 28800 s .

Vogliamo analizzare la produzione di una settimana, ma poiché la nostra simulazione parte da uno stato iniziale in cui il sistema è fermo, la produzione iniziale sarà inferiore a quella che si ha a regime, così come diverso sarà il flusso interno di produzione sulla linea; per far sì che le simulazioni non siano falsate dal *tempo di warm-up* del sistema, anziché 5 giorni simuliamo su 7 giorni.

Le giornate in termini di secondi sono (ciascuna dura 28800, per cui vi è una tale differenza in secondi tra ciascuna delle 7 giornate), idealmente:

Giornata	range temporale [s]
1	0-28800
2	28800-57600
3	57600-86400
4	86400-115200
5	115200-144000
6	144000-172800
7	172800-201600

TABELLA 5.1 RAPPRESENTAZIONE DELLE GIORNATE DI LAVORO

Nel seguito riportiamo i risultati delle simulazioni, diamo eventualmente delle spiegazioni su come le abbiamo eseguite e sui dati stessi riportati, e li analizziamo.

Dobbiamo preliminarmente specificare che per passare da una simulazione ad un'altra, -ossia per cambiare configurazione-, bisogna effettuare delle modifiche nel codice che si fa girare.
Seppur non abbiamo annesso tutti i codici negli allegati in quanto sarebbero stati molti e simili, ed evitiamo di spiegare tutte le differenze, - solamente tecniche e minime-, tra un codice per una configurazione ed un altro, bisogna tenere presente che alcune modifiche sono state effettuate nei codici prima di simulare.
Le modifiche alla configurazione possono essere di due tipi:

1. Modifica del numero di operatori standard ammessi;
2. Modifica del numero di WIP annessi.

Nel primo caso, le modifiche da effettuare sul codice sono più leggere, -si tratta per lo più di modificare valori-, mentre nel secondo, alcune parti di codice vanno proprio eliminate/aggiunte.

5.1 Simulazione 1

Partiamo dalla situazione descritta finora, che abbiamo assunto essere quella di partenza, ossia quella che prevede in produzione 9 operatori standard ed un saldatore, e 5 jobs autorizzati potenzialmente entro il sistema.
Riportiamo dapprima un aggregato di dati che spiega l'evoluzione del sistema, sotto questa configurazione (nelle successive simulazioni la configurazione sarà modificata, ma non strutturalmente: le modifiche saranno dovute al fatto che alcuni task non potranno essere assegnati per mancanza di operatori disponibili o per un minor numero di WIP consentito; nelle simulazioni successive infatti, andremo in difetto su numero di operatori e WIP, mai in eccesso); lo troviamo nella successiva tabella 5.1.
Per questa prima simulazione andiamo a riportare quasi tutte le elaborazioni sui dati che sono state fatte, in modo da dare una panoramica completa di ciò che è stato fatto e come, mentre andando avanti nel mostrare i risultati delle simulazioni, riporteremo solo i dati e gli output principali, rilevanti in vista del confronto tra scenari: sarebbe inutilmente lungo e piuttosto ripetitivo riportare tutte le tabelle.

1 Job + 4Wip

Index	incorso	daassegnareconprio			assegnatoconprio			finisce	
1	[]	22	16	26	22	16	26	22	0
2	[16,26]	4	0	0	4	0	0	26	0
3	[4,16]	7	0	0	7	0	0	16	0
4	[4,7]	1	0	0	1	0	0	7	0
5	[1,4]	27	0	0	27	0	0	27	0
6	[1,4]	9	26	0	9	26	0	26	0
7	[1,4,9]	7	0	0	7	0	0	1	0
8	[4,7,9]	17	0	0	17	0	0	7	0
9	[4,9,17]	0	0	0	0	0	0	17	0
10	[4,9]	2	0	0	2	0	0	9	0
11	[2,4]	8	0	0	8	0	0	4	0
12	[2,8]	23	0	0	23	0	0	23	0
13	[2,8]	5	22	0	5	22	0	22	0

14	[2,5,8]	4	0	0	4	0	0	8	0
15	[2,4,5]	31	0	0	31	0	0	31	0
16	[2,4,5]	27	0	0	27	0	0	27	0
17	[2,4,5]	9	26	0	9	26	0	26	0
18	[2,4,5,9]	7	0	0	7	0	0	7	0
19	[2,4,5,9]	0	0	0	0	0	0	4	0
20	[2,5,9]	0	0	0	0	0	0	2	0
21	[5,9]	19	0	0	19	0	0	19	0
22	[5,9]	3	0	0	3	0	0	9	0
23	[3,5]	8	0	0	8	0	0	5	0
24	[3,8]	24	0	0	24	0	0	24	0
25	[3,8]	6	23	0	6	23	0	23	0
26	[3,6,8]	5	22	0	5	22	0	22	0
27	[3,5,6,8]	4	0	0	4	0	0	8	0
28	[3,4,5,6]	30	0	0	30	0	0	30	0
29	[3,4,5,6]	0	0	0	0	0	0	4	0
30	[3,5,6]	0	0	0	0	0	0	5	0
31	[3,6]	0	0	0	0	0	0	3	0
32	6	21	0	0	21	0	0	21	0
33	6	0	0	0	0	0	0	6	0
34	[]	25	0	0	25	0	0	25	0
35	[]	10	24	0	10	24	0	24	0
36	10	6	23	0	6	23	0	23	0
37	[6,10]	5	22	0	5	22	0	22	0
38	[5,6,10]	4	0	0	4	0	0	4	0
39	[5,6,10]	0	0	0	0	0	0	10	0
40	[5,6]	18	0	0	18	0	0	18	0
41	[5,6]	11	16	0	11	16	0	16	0
42	[5,6,11]	1	0	0	1	0	0	1	0
43	[5,6,11]	17	0	0	17	0	0	17	0
44	[5,6,11]	2	0	0	2	0	0	5	0
45	[2,6,11]	0	0	0	0	0	0	11	0
46	[2,6]	29	0	0	29	0	0	29	0
47	[2,6]	13	0	0	13	0	0	6	0
48	[2,13]	25	0	0	25	0	0	25	0
49	[2,13]	10	24	0	10	24	0	24	0
50	[2,10,13]	6	23	0	6	23	0	23	0
51	[2,6,10,13]	5	22	0	5	22	0	22	0
52	[2,5,6,10,13]	4	0	0	4	0	0	2	0
53	[4,5,6,10,13]	19	0	0	19	0	0	19	0
54	[4,5,6,10,13]	3	0	0	3	0	0	10	0
55	[3,4,5,6,13]	18	0	0	18	0	0	4	0
56	[3,5,6,13,18]	16	0	0	16	0	0	18	0
57	[3,5,6,13,16]	11	0	0	11	0	0	16	0
58	[3,5,6,11,13]	1	0	0	1	0	0	1	0
59	[3,5,6,11,13]	17	0	0	17	0	0	17	0
60	[3,5,6,11,13]	2	0	0	2	0	0	5	0

61	[2,3,6,11,13]	0	0	0	0	0	0	11	0	
62	[2,3,6,13]	28	0	0	28	0	0	28	0	
63	[2,3,6,13]	27	0	0	27	0	0	27	0	
64	[2,3,6,13]	9	26	0	9	26	0	26	0	
65	[2,3,6,9,13]	7	0	0	7	0	0	7	0	
66	[2,3,6,9,13]	0	0	0	0	0	0	3	0	
67	[2,6,9,13]	20	0	0	20	0	0	20	0	
68	[2,6,9,13]	12	0	0	12	0	0	13	0	
69	[2,6,9,12]	15	0	0	15	0	0	9	0	
70	[2,6,12,15]	8	0	0	8	0	0	6	0	
71	[2,8,12,15]	0	0	0	0	0	0	8	0	
72	[2,12,15]	0	0	0	0	0	0	2	0	
73	[12,15]	19	0	0	19	0	0	19	0	
74	[12,15]	3	0	0	3	0	0	3	0	
75	[12,15]	0	0	0	0	0	0	12	0	
76	15	14	0	0	14	0	0	15	0	
77	14	33	0	0	33	0	0	33	0	Rilascio Job 1.
78	14	31	21	0	31	21	0	31	0	t=5095.9
79	[14,21]	25	0	0	25	0	0	21	0	
80	[14,25]	24	0	0	24	0	0	25	0	
81	[14,24]	10	23	0	10	23	0	24	0	
82	[10,14,23]	6	22	0	6	22	0	22	0	
83	[6,10,14,23]	4	0	0	4	0	0	23	0	
84	[4,6,10,14]	5	0	0	5	0	0	4	0	
85	[5,6,10,14]	0	0	0	0	0	0	10	0	
86	[5,6,14]	18	0	0	18	0	0	18	0	
87	[5,6,14]	11	16	0	11	16	0	16	0	
88	[5,6,11,14]	1	0	0	1	0	0	1	0	
89	[5,6,11,14]	17	0	0	17	0	0	17	0	
90	[5,6,11,14]	2	0	0	2	0	0	5	0	
91	[2,6,11,14]	0	0	0	0	0	0	11	0	
92	[2,6,14]	29	0	0	29	0	0	29	0	
93	[2,6,14]	13	27	0	13	27	0	27	0	
94	[2,6,13,14]	9	26	0	9	26	0	26	0	
95	[2,6,9,13,14]	7	0	0	7	0	0	7	0	
96	[2,6,9,13,14]	0	0	0	0	0	0	9	0	
97	[2,6,13,14]	8	0	0	8	0	0	6	0	
98	[2,8,13,14]	0	0	0	0	0	0	8	0	
99	[2,13,14]	0	0	0	0	0	0	2	0	
100	[13,14]	19	0	0	19	0	0	19	0	
101	[13,14]	3	0	0	3	0	0	14	0	
102	[3,13]	32	0	0	32	0	0	32	0	Rilascio Job 2.
103	[3,13]	30	0	0	30	0	0	30	0	t=6320 s
104	[3,13]	25	0	0	25	0	0	25	0	
105	[3,13]	10	24	0	10	24	0	24	0	
106	[3,10,13]	6	23	0	6	23	0	23	0	
107	[3,6,10,13]	5	22	0	5	22	0	22	0	

108	[3,5,6,10,13]	4	0	0	4	0	0	4	0	
109	[3,5,6,10,13]	0	0	0	0	0	0	10	0	
110	[3,5,6,13]	18	0	0	18	0	0	18	0	
111	[3,5,6,13]	11	16	0	11	16	0	16	0	
112	[3,5,6,11,13]	1	0	0	1	0	0	1	0	
113	[3,5,6,11,13]	17	0	0	17	0	0	5	0	
114	[3,6,11,13,17]	0	0	0	0	0	0	17	0	
115	[3,6,11,13]	2	0	0	2	0	0	3	0	
116	[2,6,11,13]	20	0	0	20	0	0	13	0	
117	[2,6,11,20]	15	0	0	15	0	0	11	0	
118	[2,6,15,20]	28	0	0	28	0	0	20	0	
119	[2,6,15,28]	27	0	0	27	0	0	27	0	
120	[2,6,15,28]	9	26	0	9	26	0	28	0	
121	[2,6,9,15,26]	12	0	0	12	0	0	26	0	
122	[2,6,9,12,15]	7	0	0	7	0	0	7	0	
123	[2,6,9,12,15]	0	0	0	0	0	0	9	0	
124	[2,6,12,15]	8	0	0	8	0	0	6	0	
125	[2,8,12,15]	0	0	0	0	0	0	8	0	
126	[2,12,15]	0	0	0	0	0	0	2	0	
127	[12,15]	19	0	0	19	0	0	19	0	
128	[12,15]	3	0	0	3	0	0	3	0	
129	[12,15]	0	0	0	0	0	0	12	0	
130	15	14	0	0	14	0	0	15	0	
131	14	33	0	0	33	0	0	33	0	Rilascio Job 3
132	14	21	31	0	21	31	0	31	0	t=8208,8s

TABELLA 5.2 ESTRATTO INIZIALE EVOLUZIONE SISTEMA MATLAB

Questa tabella è stata disegnata per avere una visione compatta del comportamento del sistema, per lo meno nel periodo iniziale; ne abbiamo riportato i valori sino allo stato 132, al quale corrisponde l'uscita dal sistema del terzo job, -come indicato-.

Gli eventuali colori di rappresentazione dei job, e del relativo rilascio, nella tabella sono stati inseriti per una migliore comprensione e rappresentazione della differenza tra un job e l'altro, in modo che fosse anche più agevole nel leggere la tabella, identificare a quale job sia riferito il task in assegnazione, o conclusione, ecc. e poter seguire così l'andamento del processamento dei vari job. Questa rappresentazione a colori è utile anche per osservare e verificare il rispetto delle priorità dei job nelle assegnazioni.

Nella successiva tabella 5.3 riportiamo la colorazione eventuale per i job: riferirsi a questa tabella per interpretare eventuali colorazioni dei numeri negli output delle simulazioni che riportiamo:

Colore	
nero	job 1
fucsia	job 2
blu chiaro	job 3
arancione	job 4
verde	job 5
viola	job 6

pervinca	job 7
blu	job 8

TABELLA 5.3 RAPPRESENTAZIONE DEI COLORI DEI JOB

La successiva tabella 5.4 riporta i risultati della variabile *out* della simulazione, che, come abbiamo visto, memorizza i tempi in cui ciascun job dal primo all'ultimo (145-esimo) è completato e quindi ritenuto espulso dal sistema (o comunque lo riteniamo da lì in avanti estraneo alla produzione). La prima colonna contiene i valori della variabile *out* esattamente come riportati da Matlab; la seconda colonna contiene gli stessi dati della prima ma espressi in modo tale da essere la corretta rappresentazione in secondi, e l'ultima la differenza temporale intercorsa tra un completamento e il successivo.

Job Out	Jobs Out	t (job out) [s]	Δt [s]
1	509.590.000.000.000	5095,9	0
2	632.000.000.000.000	6320	1224,1
3	820.880.000.000.000	8208,8	1888,8
4	940.250.000.000.000	9402,5	1193,7
5	111.460.000.000.000	11146	1743,5
6	123.867.000.000.000	12386,7	1240,7
7	141.572.000.000.000	14157,2	1770,5
8	154.340.000.000.000	15434	1276,8
9	178.917.000.000.000	18.104	2670
10	188.786.000.000.000	19482	1378
11	208.877.000.000.000	20887,7	1405,7
12	22.221	22221	1333,3
13	239.731.000.000.000	23973,1	1752,1
14	253.027.000.000.000	26395	2421,9
15	268.471.000.000.000	26847,1	452,1
16	283.454.000.000.000	28345,4	1498,3
17	298.586.000.000.000	29858,6	1513,2
18	313.920.000.000.000	31392	1533,4
19	328.592.000.000.000	32859,2	1467,2
20	343.742.000.000.000	34374,2	1515
21	359.081.000.000.000	35908,1	1533,9
22	373.588.999.999.999	37358,9	1450,8
23	389.487.000.000.000	38948,7	1589,8
24	402.644.000.000.000	40264,4	1315,7
25	420.233.000.000.000	42023,3	1758,9
26	432.968.000.000.000	43296,8	1273,5
27	450.682.999.999.999	45068,3	1771,5
28	463.833.000.000.000	46383,3	1315
29	480.652.000.000.000	48065,2	1681,9
30	493.245.000.000.000	49324,5	1259,3
31	510.636.000.000.000	51063,6	1739,1
32	524.165.000.000.000	52416,5	1352,9

33	541.412.000.000.000	54141,2	1724,7
34	555.069.000.000.000	55506,9	1365,7
35	571.910.000.000.000	57191	1684,1
36	584.246.000.000.000	58424,6	1233,6
37	601.395.000.000.000	60139,5	1714,9
38	615.729.000.000.000	61572,9	1433,4
39	630.679.000.000.000	63067,9	1495
40	646.104.000.000.001	64610,4	1542,5
41	660.982.000.000.001	66098,2	1487,8
42	676.502.000.000.000	67650,2	1552
43	693.110.000.000.000	69311	1660,8
44	707.391.000.000.000	70739,1	1428,1
45	722.356.000.000.000	72235,6	1496,5
46	739.026.000.000.000	73902,6	1667
47	752.056.000.000.000	75205,6	1303
48	769.367.000.000.000	76936,7	1731,1
49	783.435.000.000.000	78343,5	1406,8
50	800.502.000.000.000	80050,2	1706,7
51	813.656.000.000.000	81365,6	1315,4
52	830.982.000.000.000	83098,2	1732,6
53	843.670.000.000.000	84367	1268,8
54	861.117.000.000.000	86111,7	1744,7
55	873.503.000.000.000	87350,3	1238,6
56	891.932.000.000.000	89193,2	1842,9
57	902.030.000.000.000	90203	1009,8
58	922.988.000.000.001	92298,8	2095,8
59	933.189.000.000.001	93318,9	1020,1
60	952.903.000.000.001	95290,3	1971,4
61	965.094.000.000.000	96509,4	1219,1
62	982.349.000.000.001	98234,9	1725,5
63	995.564.000.000.001	995564	897329,1
64	101.247.800.000.000	101247,8	-894316
65	102.636.300.000.000	102636,3	1388,5
66	104.223.500.000.000	104223,5	1587,2
67	105.681.400.000.000	105681,4	1457,9
68	107.364.000.000.000	107364	1682,6
69	108.642.500.000.000	108642,5	1278,5
70	110.337.600.000.000	110337,6	1695,1
71	111.643.100.000.000	111643,1	1305,5
72	113.332.700.000.000	113332,7	1689,6
73	114.622.000.000.000	114622	1289,3
74	116.211.100.000.000	116211,1	1589,1
75	117.612.900.000.000	117612,9	1401,8
76	119.417.700.000.000	119417,7	1804,8
77	120.619.300.000.000	120619,3	1201,6
78	123.106.900.000.000	123106,9	2487,6
79	124.238.600.000.000	124238,6	1131,7

80	126.068.100.000.000	126068,1	1829,5
81	127.480.500.000.000	127480,5	1412,4
82	129.169.500.000.000	129169,5	1689
83	130.554.200.000.000	130554,2	1384,7
84	132.146.900.000.000	132146,9	1592,7
85	133.550.800.000.000	133550,8	1403,9
86	135.204.900.000.000	135204,9	1654,1
87	136.457.300.000.000	136457,3	1252,4
88	138.084.400.000.000	138084,4	1627,1
89	139.371.000.000.000	139371	1286,6
90	141.012.000.000.000	141012	1641
91	142.400.200.000.000	142400,2	1388,2
92	144.054.000.000.000	144054	1653,8
93	145.282.700.000.000	145282,7	1228,7
94	147.052.100.000.000	147052,1	1769,4
95	148.381.400.000.000	148381,4	1329,3
96	149.996.100.000.000	149996,1	1614,7
97	151.385.000.000.000	151385	1388,9
98	152.932.100.000.000	152932,1	1547,1
99	154.465.400.000.000	154465,4	1533,3
100	155.922.000.000.000	155922	1456,6
101	157.487.700.000.000	157487,7	1565,7
102	158.929.700.000.000	158929,7	1442
103	160.454.200.000.000	160454,2	1524,5
104	161.923.400.000.000	161923,4	1469,2
105	163.438.100.000.000	163438,1	1514,7
106	164.884.900.000.000	164884,9	1446,8
107	166.384.900.000.000	166384,9	1500
108	167.952.000.000.000	167952	1567,1
109	169.283.900.000.000	169283,9	1331,9
110	170.992.300.000.000	170992,3	1708,4
111	172.323.100.000.000	172323,1	1330,8
112	174.066.300.000.000	174066,3	1743,2
113	175.459.700.000.000	175459,7	1393,4
114	177.033.500.000.000	177033,5	1573,8
115	178.427.600.000.000	178427,6	1394,1
116	180.037.300.000.000	180037,3	1609,7
117	181.493.700.000.000	181493,7	1456,4
118	182.961.200.000.000	182961,2	1467,5
119	184.485.600.000.000	184485,6	1524,4
120	185.902.900.000.000	185902,9	1417,3
121	187.441.500.000.000	187441,5	1538,6
122	188.929.100.000.000	188929,1	1487,6
123	190.432.100.000.000	190432,1	1503
124	191.955.599.999.999	191955,6	1523,5
125	193.445.000.000.000	193445	1489,4
126	194.928.600.000.000	194928,6	1483,6

127	196.461.300.000.000	196461,3	1532,7
128	198.012.100.000.000	198012,1	1550,8
129	199.540.700.000.000	199540,7	1528,6
130	201.137.900.000.000	201137,9	1597,2
131	202.651.700.000.000	202651,7	1513,8
132	204.140.900.000.000	204140,9	1489,2
133	205.560.700.000.000	205560,7	1419,8
134	207.276.600.000.000	207276,6	1715,9
135	208.651.000.000.000	208651	1374,4
136	210.276.100.000.000	210276,1	1625,1
137	211.614.000.000.000	211614	1337,9
138	213.329.400.000.000	213329,4	1715,4
139	214.603.299.999.999	214603,3	1273,9
140	216.409.099.999.999	216409,1	1805,8
141	217.672.799.999.999	217672,8	1263,7
142	219.418.100.000.000	219418,1	1745,3
143	220.716.300.000.000	220716,3	1298,2
144	222.473.099.999.999	222473,1	1756,8
145	223.779.599.999.999	223779,6	1306,5

TABELLA 5.4 COMPLETAMENTO JOBS SIMULAZIONE 1

Nella successiva tabella 5.5, è riportata la determinazione che abbiamo effettuato per le giornate della simulazione 1, in base al relativo output dei tempi. Nel determinare la fine di ciascuna giornata, facendo sì che la scelta si avvicini il più possibile ai valori ideali (quelli di tabella 5.1), -per questo viene effettuata l'analisi degli scostamenti rispetto alle giornate ideali-, viene inoltre controllato che nello stato *i* scelto come termine di una giornata non vi siano transport task in corso (altrimenti si effettua una scelta differente). Questo viene fatto perché si suppone che un *mt* che risulti ancora in corso a fine giornata possa venire interrotto e ripreso dal punto di interruzione alla ripresa della produzione il giorno successivo, mentre un *transport task* deve necessariamente terminare il suo corso. Facciamo notare che i *transport task* sono molto brevi (le durate variano in un range di 5-25 s), per cui è molto probabile che nello stato scelto non vi siano *tt* in corso, altrimenti si sceglie un istante successivo o precedente, sempre facendo in modo nel complesso della scelta delle giornate, che gli scostamenti cumulati rispetto alle giornate ideali siano il più piccoli possibile.

Dunque, nella determinazione degli estremi temporali di ciascuna giornata simulata, si controlla quali task risultano in corso (si usa il comando `find(S(i,1:33))`, con *i* impostato sullo stato candidato ad essere il termine della giornata perché vicino al termine della giornata ideale) e ci si assicura che per lo stato scelto i task in corso siano, al massimo, tutti di tipo *mt* (quindi compresi tra 1 e 15, non superiori a 15). Questo check viene fatto nel determinare la successiva matrice 5.5, ma l'indicazione dei task in corso nell'istante di fine di ciascuna giornata lo troviamo nella tabella ancora successiva, la 5.6, nella terza colonna.

Calcoli

Fine giornata	i	t(i)	scostamento rispetto a giornate ideali	scostamento totale rispetto a giornate ideali		
1	491	28792,2	-7,80			
2	1002	57623,3	23,30	15,5		
3	1514	86.286,2	-113,8	-98,3		
4	2039	115261,9	61,9	-36,4		
5	2531	144.039,7	39,7	3,3		
6	3054	172.791,5	-9	-5,2		
7	3570	201.601,1	1	-4	[s]	Scostamento accettabilissimo

TABELLA 5.5 DETERMINAZIONE GIORNATE SIMULAZIONE 1

In base ai calcoli precedenti, incrociando i dati con quelli di tabella 5.4, troviamo i seguenti dati di output per la simulazione 1:

Giornata	i	t	task in corso a fine giornata	Completati	Δjobs		
1	1:491	0:28792,2	5,6,15	16,749	16,749		
2	491:1002	t1:57623,3	5,6,10,14	35,749	19		
3	1002:1514	t2:86286,2	4,5,6,10,15	55,749	20	-> interrompo produzione giornaliera	sono tutti MT (no transport) allora ok (suppongo poter lasciare i job e riprenderli il giorno dopo)
4	1514:2039	t3:115261,9	2,6,9,13,14	73,749	18		
5	2039:2531	t4:144039,7	3,13	91,9969	18,248		
6	2531:3054	t5:172791,5	3,5,6,11,12	111,5062	19,509		
7	3054:3570	t6:201601,1	5,6,11,15	130,749	19,243		

TABELLA 5.6 OUTPUT SIMULAZIONE 1

L'utilizzo e il tasso di utilizzo degli operatori in questa situazione di Simulazione 1 sono dati nella seguente tabella:

	Utilizzo (per lavoratore per giornata) [s]									
	1	2	3	4	5	6	7	8	9	10
1	26580,8	23684,8	20785,2	17671,4	20592,1	17258,4	17964,8	15105,3	9877,5	18326,4
2	23156,3	25355	24810,1	21619,8	19457,3	20216,6	20278,5	13844,2	14634,2	19552,4
3	27066	25077,2	24842,1	21966,1	23715	17455,5	15937,9	14925,2	12351,6	19698,2
4	25725	22571,5	23616,7	20360,5	22768,8	21996,5	19560,4	15349,1	13159,7	20233,7
5	25013,8	24953,7	23613,2	20598,9	20873,9	20195,7	17490,7	12493,5	12073,1	19022,4
6	26380,8	26193,2	20263,2	21560,1	23444,7	20139,9	22504,3	12502,6	11115,1	20091,9
7	26422	26202,8	22627,3	18956,4	20706,9	21257,9	20217,9	13131,9	14171,3	19636,2

	Worker utilization rate (per giornata) [%]									
1	0,9232	0,8226	0,7219	0,6138	0,7152	0,5994	0,6239	0,5246	0,3431	0,6365
2	0,8038	0,8801	0,8612	0,7504	0,6754	0,7017	0,7039	0,4805	0,5080	0,6787
3	0,9453	0,8759	0,8677	0,7672	0,8283	0,6097	0,5567	0,5213	0,4314	0,688

4	0,8946	0,7849	0,8212	0,708	0,7918	0,7649	0,6802	0,5337	0,4576	0,7036
5	0,8777	0,8756	0,8285	0,7228	0,7324	0,7086	0,6137	0,4384	0,4236	0,6675
6	0,9180	0,9115	0,7051	0,7502	0,8158	0,7008	0,7831	0,4351	0,3868	0,6992
7	0,9184	0,9108	0,7865	0,6589	0,7198	0,7389	0,7028	0,4565	0,4926	0,6826

TABELLA 5.7 UTILIZZO E TASSO DI UTILIZZO OPERATORI SIMULAZIONE 1

I dati su utilizzo e tasso di utilizzo sono ottenuti dopo aver simulato, usando il seguente codice, nel quale abbiamo inserito manualmente gli estremi di ciascuna giornata, trovati in precedenza:

```
wur=zeros(7,10); %tasso di utilizzo ("worker utilization rate") di ciascuno dei 10 lavoratori, per giornata
u=zeros(7,10); %utilizzo worker (riga=giornata - colonna=worker)
giornata=[1 434;435 945;946 1457;1458 1981;1982 2494;2495 3008;3009 3525]; %estremi temporali delle giornate di lavoro secondo questo codice (servono per le statistiche sull'utilizzo

for i=1:7 %giornata
    for w=1:10 %lavoratore
        for j=giornata(i,1):giornata(i,2)
            if assegnato(j,w)~=0 || W(j,w)~=0
                u(i,w)=u(i,w)+ttemp(j);
            end
        end
        wur(i,w)=u(i,w)/(t(giornata(i,2))-t(giornata(i,1)));
    end
end
```

5.2 Simulazione 2

In questa simulazione andiamo ad autorizzare un job in meno dentro il sistema; abbiamo quindi la seguente impostazione: 9 operatori standard e 1 saldatore e 4 jobs (1 job principale + 3 wip) ammessi nel sistema.

La situazione è simile alla precedente; l'output diminuisce. La successiva tabella riporta le giornate simulate e l'output risultante, per questa configurazione:

Giornata	i	t	S	Completati	Δjobs		
1	1:434	0:28801,6	3,5,6,11,13	14	14		
2	434:945	t1:57603,1	6,10,14	33	19		
3	945:1457	t2:86378	6,10,15	52,5062	19	-> interrompo produzione giornaliera	sono tutti MT (no transport) allora ok, suppongo poter lasciare i job e riprenderli il giorno dopo)
4	1457:1981	t386378	2,5,6,13,14	71,5062	19		
5	1981:2494	t4:143909	2,12,15	90,749	19,2428		
6	2494:3008	t5:172832,4	3,13	109,9969	19,2479		
7	3008:3525	t6:201546,1	3,4,10,12	129,5062	19,5093		

TABELLA 5.8 OUTPUT SIMULAZIONE 2

La successiva invece riporta il tasso di utilizzo dei lavoratori, per questa configurazione:

	Worker utilization rate (per giornata) [%]									
1	0,8446	0,7765	0,6660	0,5993	0,5903	0,5114	0,5059	0,482	0,2186	0,5708
2	0,8586	0,8289	0,7979	0,8167	0,7954	0,8229	0,7670	0,3429	0,3450	0,6710
3	0,8985	0,8511	0,8265	0,7797	0,7981	0,6011	0,5340	0,7535	0,3502	0,6880
4	0,8606	0,8071	0,8061	0,7830	0,8108	0,7771	0,8161	0,3803	0,3970	0,6930
5	0,9376	0,8614	0,7908	0,7945	0,7838	0,7249	0,7953	0,3464	0,3779	0,6972
6	0,8602	0,8563	0,8046	0,8206	0,8133	0,5907	0,5129	0,7879	0,3460	0,6802
7	0,8891	0,8700	0,7957	0,7949	0,8114	0,6135	0,5322	0,8119	0,3129	0,6962

TABELLA 5.9 TASSO DI UTILIZZO OPERATORI SIMULAZIONE 2

5.3 Simulazioni 3 e 4

Andiamo adesso a diminuire il numero degli operatori (standard) disponibili da 9 a 8, lasciando il numero dei WIP consentiti entro il processo a 3.

In questo caso il calcolo delle giornate è:

	Calcoli				
503	28920,5	120,5	-0,4		
1022	57479,1	-120,9			
1542	86405,7	5,7	5,3		
2061	115202,4	2,4	7,7		
2574	144.025,4	25,4	33,1		
3092	172.672,1	-127,9	-94,8		
3607	201.698,1	98,1	3,3	[s]	ok

TABELLA 5.10 DETERMINAZIONE GIORNATE SIMULAZIONE 3

E l'output della simulazione è:

Giornata	i	t	S	Completati	Δjobs		
1	1:503	0:28920,5	3,12,15	16,749	16,749		
2	503:1022	t1:57479,1	1,3,6,13	36,5062	19,7572		
3	1022:1542	t2:86405,7	5,6,14	55,749	19,2428	-> interrompo produzione giornaliera	sono tutti MT (no transport) allora ok (suppongo poter lasciare i job e riprenderli il giorno dopo)
4	1542:2061	t3:115202,4	2,5,15	74,749	19		
5	2061:2574	t4:144025,4	2,6,13,14	93,749	19		
6	2574:3092	t5:172672,1	3,9,12,15	112,749	19		
7	3092:3607	t6:201698,1	3,6,10,13	132,5062	19,7572		

TABELLA 5.11 OUTPUT SIMULAZIONE 3

Notiamo subito che l'output di questa configurazione è migliore di quello di entrambe le configurazioni precedenti, che supponevano l'impiego di 1 operatore in più.

Il tasso di utilizzo di ciascun lavoratore per questa configurazione risulta:

Worker utilization rate (per giornata) [%]

1	0,8210	0,8350	0,7635	0,7319	0,7623	0,7488	0,6987	0,6782	0,6604
2	0,7769	0,8061	0,8631	0,8702	0,8760	0,7826	0,7637	0,7268	0,6921
3	0,7764	0,8213	0,8663	0,7988	0,8420	0,8151	0,7532	0,6783	0,6927
4	0,8921	0,8456	0,8707	0,7729	0,8303	0,8003	0,7293	0,7172	0,6875
5	0,8237	0,7854	0,7958	0,8449	0,7862	0,7948	0,7966	0,7422	0,6798
6	0,8197	0,8238	0,6828	0,8186	0,7653	0,8333	0,8404	0,7237	0,6979
7	0,7899	0,8526	0,7721	0,9231	0,7707	0,7689	0,7994	0,7697	0,6895

TABELLA 5.12 TASSO DI UTILIZZO OPERATORI SIMULAZIONE 3

Riportiamo adesso il numero dei WIP consentiti a 4, come nel caso di partenza, per verificare se l'output salga ulteriormente. L'output risultante dalla simulazione con questa configurazione (**Simulazione 4**) è il seguente:

Giornata	i	t	S	Completati	Δjobs		
1	1:508	0:28803,5	3,6,12,15	16,749	16,749		
2	508:1016	t1:57600,1	6,10,13,14	35,749	19		
3	1016:1533	t2:86382,6	3,6,11,12,15	54,749	19	-> interrompo produzione giornaliera	sono tutti MT (no transport) allora ok (suppongo poter lasciare i job e riprenderli il giorno dopo)
4	1533:2052	t3:115125,9	3,5,6,10,13	74,5062	19,7572		
5	2052:2568	t4:144069,9	6,10,14	93,749	19,2428		
6	2568:3064	t5:172820,7	2,13,14	111,749	18		
7	3064:3582	t6:201577,6	2,12,15	130,749	19		

TABELLA 5.13 OUTPUT SIMULAZIONE 4

Vediamo che l'output totale (i pezzi completati nel tempo della simulazione ossia una settimana intesa come 7 giorni, in ciascuna tabella di output evidenziato con un colore) non è ulteriormente aumentato bensì diminuito.

Questa situazione ci porta a pensare che la produzione migliore si abbia in corrispondenza della configurazione 3, proprio a causa di questo picco nella produzione settimanale a confronto di configurazioni.

Il tasso di utilizzo dei lavoratori per questa configurazione è:

	Worker utilization rate (per giornata) [%]								
1	0,8210	0,8350	0,7635	0,7319	0,7623	0,7488	0,6987	0,6782	0,6604
2	0,7769	0,8061	0,8631	0,8702	0,8760	0,7826	0,7637	0,7268	0,6921
3	0,7764	0,8213	0,8663	0,7988	0,8420	0,8151	0,7532	0,6783	0,6927
4	0,8921	0,8456	0,8707	0,7729	0,8303	0,8003	0,7293	0,7172	0,6875
5	0,8237	0,7854	0,7958	0,8449	0,7862	0,7948	0,7966	0,7422	0,6798
6	0,8197	0,8238	0,6828	0,8186	0,7653	0,8333	0,8404	0,7237	0,6979
7	0,7899	0,8526	0,7721	0,9231	0,7707	0,7689	0,7994	0,7697	0,6895

TABELLA 5.14 TASSO DI UTILIZZO OPERATORI SIMULAZIONE 3

5.4 Simulazione 5

Andiamo a diminuire ulteriormente di 1 il numero degli operatori (standard) disponibili: da 8 a 7, lasciando ancora il numero di WIP consentiti entro il processo a 3.
I risultati che otteniamo sono i seguenti:

Giornata	i	t	S	Completati	Δjobs	
1	1:485	0:28834,2	4,6,10,13	16,5062	16,5062	
2	485:959	t1:57566,1	6,10,12,15	33,749	17,2428	
3	959:1445	t2:86299,1	3,6,12,15	51,749	18	-> interrompo produzione giornaliera
4	1445:1902	t3:115277,7	3,10,12,15	68,749	17	
5	1902:2373	t4:143917,9	4,6,10	86,749	18	
6	2373:2859	t5:172922	6,10,12	104,5062	17,7572	
7	2859:3334	t6:201577,4	3,6,10,12	121,9969	17,4907	

TABELLA 5.15 OUTPUT SIMULAZIONE 5

Vediamo che l'output è notevolmente diminuito rispetto al caso precedente, che per finora risulta migliore. Per quanto riguarda il tasso di utilizzo degli operatori per ciascuna giornata, i dati sono riportati nella successiva tabella 5.14.

	Worker utilization rate (per giornata) [%]							
1	0,8981	0,8651	0,7567	0,8236	0,8830	0,8111	0,7862	0,6404
2	0,8493	0,8494	0,8423	0,8569	0,8272	0,8533	0,8845	0,6384
3	0,8254	0,9040	0,8737	0,8552	0,9323	0,9068	0,8676	0,6529
4	0,9247	0,7997	0,8047	0,7924	0,8105	0,7323	0,7533	0,5989
5	0,8621	0,8693	0,8725	0,8063	0,8323	0,7729	0,8382	0,6457
6	0,8327	0,9053	0,8381	0,8668	0,8746	0,8754	0,8379	0,6466
7	0,8511	0,8515	0,8485	0,8788	0,842	0,8554	0,8406	0,6323

TABELLA 5.16 TASSO DI UTILIZZO OPERATORI SIMULAZIONE 5

5.5 Confronto risultati e scelta della configurazione

La successiva tabella confronta i risultati delle simulazioni. Essa contiene delle statistiche extra rispetto alle tabelle precedenti, ossia il tasso di utilizzo (*worker utilization rate*) totale (cioè su tutti i 7 giorni) per il saldatore e quello totale medio degli operatori standard.

Sym N.	N. operatori	N. wip	Output	Tasso di utilizzo totale medio operatori standard	Tasso di utilizzo totale saldatore
1	9	4	130,749	0,6980	0,6774
2	9	3	129,5062	0,6921	0,6702

3	8	3	132,5062	0,7967	0,6857
4	8	4	130,749	0,7953	0,6813
5	7	3	121,9969	0,8405	0,6324

TABELLA 5.17 CONFRONTO RISULTATI SIMULAZIONI

Dal confronto delle situazioni simulate, emerge che la situazione migliore è quella in cui sono impiegati 8 operatori standard, -oltre al saldatore-, e sono consentiti 3 WIP entro il processo.

Rispetto alla situazione di partenza (Sym n.1) vediamo che diminuire il numero di WIP consentiti entro il processo anche solo di 1 unità fa diminuire tutte le misure di performance analizzate, per cui non avrebbe un senso farlo.

La configurazione della simulazione n.3, -che vediamo evidenziata-, risulta la migliore in assoluto: l'output dei primi 7 giorni aumenta di 1.7572 unità allo stesso tempo in cui viene ridotto di 1 il numero degli operatori standard. Poiché l'inizio della simulazione, -come abbiamo spiegato anche precedentemente, ma è importante rimarcarlo-, è una condizione diversa e particolare rispetto al resto dei giorni, nei quali il sistema si trova non nella situazione iniziale di transizione ma a regime, l'aumento nell'output in 7 giorni a regime sarà anche superiore a 1.7572 unità.

Ad ogni modo, i risultati ci dicono che l'azienda, scegliendo la configurazione di Sym n.3 rispetto a quella attuale, avrebbe un risparmio di costi pari, al minimo, al costo di un operatore assunto ed un aumento di produttività, con relativo aumento in termini economici,- quindi di fatturato o di utile, o qualsiasi misura economica si voglia considerare-.

Abbiamo effettuato una stima dell'aumento in produzione che si avrebbe su un anno di lavoro; per farlo abbiamo considerato i dati ottenuti sui 6 giorni, tolto il primo in cui la situazione è anomala e quindi il dato di output falsato dall'esigenza per il sistema di "scaldarsi" (*warm up*).

Abbiamo considerato un anno di lavoro in settimane da 5 giorni, usando i dati ottenuti sui 6 giorni (abbiamo effettuato i necessari aggiustamenti); non abbiamo considerato eventuali festività e stop dell'impianto in quanto non conosciamo con esattezza i giorni lavorativi polacchi, né le specifiche esigenze di chiusura dell'azienda. Ad ogni modo la stima è precisa, se ben interpretata, ossia: stima l'aumento in produzione con la nuova configurazione per un anno di produzione, con impianto attivo 5 giorni su 7 per 8 ore (maggiori dettagli sono dati nel seguito, insieme alla stima). Troviamo i dettagli nella successiva tabella 5.18:

Aumento produzione annuale rispetto a caso corrente senza transitorio (6gg)
(Non considero il primo giorno perché anomalo)

	Produzione su 6gg (escluso il primo)
Sym 1	114
Sym 3	115,7572

Extra produzione = 1,7572

1y=52,1429 w

gg lavoro= 52,1429*5=260,7145
n. settimane da 6 giorni=260,7145/6=43,4524

Produzione attesa in 52,1429 w= 43,4524*115,7572=	5029,93 u
Produzione corrente in 52,1429 w=43,4524*114=	4953,57 u
Extra produzione attesa in 52,1429 w= 43,4524*1,7572=	76,3546 u
Aumento percentuale atteso=	15,42%

TABELLA 5.18 AUMENTO PRODUZIONE ANNUALE ATTESO

Purtroppo non è possibile fare una stima economica di guadagno sia in termini di risparmio che di guadagno effettivo (extra guadagno), in quanto non abbiamo i dati economici sul prodotto (prezzo / margine / …) né sui costi aziendali (costo della manodopera / costi fissi / costi variabili di prodotto / …). Ma ribadiamo che vi sarà al minimo un guadagno economico dato dalla maggiore produzione e un risparmio economico dato dall'impiegare un operatore in meno.

Va fatto notare anche che, seppur effettuando questa modifica nella configurazione i tassi di utilizzo degli operatori aumentino, i valori rimangono più che accettabili. Come possiamo vedere dalla tabella 5.17, il tasso di utilizzo del saldatore aumenta di poco: dal 67.74% al 68.58%, mentre il tasso di utilizzo totale medio passa dal 69.80% per i 9 operatori della configurazione di partenza al 79.67% per gli 8 operatori della nuova configurazione. Ma vediamo meglio questo punto importante:

-79.67% di tasso di utilizzo medio per gli operatori standard significa che mediamente su 8 ore sono impegnati in effettivo lavoro 8*0.7967=6.3736 h al giorno, che significa che mediamente hanno 8-6.3736=1.6264 h, cioè 97.584 minuti in cui non stanno effettivamente lavorando, e quindi possono prendere delle pause, riposare, ecc;

- per il saldatore, un tasso di utilizzo medio del 68.58% significa che mediamente, egli, su 8 ore, sarà impegnato effettivamente in lavoro in stazione di saldatura (5/6 in stazioni Matlab) per 8*0.6858=5.4864 h, cioè avrà mediamente 8-5.4864=2.5136h=150.816 minuti di lavoro inattivo.

Facciamo notare che questo calcolo del tempo di inattività degli operatori non tiene conto della pausa pranzo che è esclusa dal calcolo in quanto sempre garantita, e ricade al di fuori delle ore di lavoro.

Queste valutazioni e annotazioni sulle tempistiche di utilizzo dei lavoratori, servivano per evidenziare che l'aumento nel tasso di utilizzo degli operatori col cambio suggerito di configurazione:

1. Nel caso del saldatore è minimo, con un aumento medio di 0.83 punti percentuali, che si traduce in 4 minuti in più di occupazione media su 8 ore;
2. Dal punto di vista dell'azienda è un cambiamento positivo, in quanto le risorse sono maggiormente sfruttate, ma lo sono entro limiti consoni al fatto che si tratta di risorse umane;
3. Non preclude il fatto che gli operatori standard continuano ad avere una buona quantità di tempo di inattività di modo che il lavoro non sia eccessivamente oneroso.

Tornando alla valutazione delle simulazioni, - continuiamo a guardare la tabella 5.17-: aumentare il numero di WIP consentiti nel processo, mantenendo il numero degli operatori standard ad 8, riporta l'output alla stessa identica situazione della condizione iniziale (output dei primi 7 giorni pari a 130.749 unità). In questo caso gli operatori sono maggiormente utilizzati, -come possiamo vedere-, dunque vi è un aumento del tasso di utilizzo degli operatori, con parità di output. Rispetto alla configurazione con 8 operatori allo stesso modo, ma 3 WIP, -che abbiamo scelto come ideale-, l'output, come detto e come ovvio, diminuisce, mentre il tasso di utilizzo degli operatori rimane pressoché invariato, per tutti.

Nell'ultima configurazione considerata, il numero di operatori standard è ridotto ancora di 1, -quindi portato a 7-, mentre il numero di WIP consentiti entro il processo resta a 3 del caso identificato come migliore tra le configurazioni precedenti. Non siamo andati a simulare oltre questa configurazione perché già con questa l'output scende di molto rispetto alle diminuzioni precedenti: vi è con questa configurazione una diminuzione di 8.7521 unità rispetto alla configurazione originale e di 10.5093 unità rispetto alla configurazione di Simulazione 3. Il tasso di utilizzo del saldatore è il più basso in assoluto con questa configurazione rispetto a tutte le altre, mentre quello medio degli operatori standard il più alto in assoluto. L'interpretazione dei dati ci suggerisce che la numero 5 sia la prima configurazione da scartare.

Capitolo 6: Modello per il Reinforcement Learning

In questo capitolo andiamo a valutare l'applicabilità del *Reinforcement Learning* (RL) per l'assegnazione degli operatori ai task nel sistema reale che abbiamo in esame.

Come dedotto dai capitoli 1 e 2, RL si può applicare principalmente a quei sistemi che sono rappresentabili mediante un *Markov Decision Process* (MDP) e caratteristica imprescindibile negli MDP è la proprietà dell'assenza di memoria per ciascuno degli stati che lo compongono.

Nel rappresentare il nostro sistema come MDP, dobbiamo dunque assicurarci che venga rispettata la proprietà di Markov (vedi cap.1). Questo significa che dobbiamo dare un'espressione dello stato che consenta di determinare esattamente dove ci si trova nel processo e in quale stato si può giungere da lì in poi, ma solamente in base alle informazioni dello stato corrente, non di quelli precedenti e quindi indipendentemente dalla storia, ossia da come si sia giunti in un particolare stato.

Il sistema in esame, purtroppo, non gode naturalmente di questa proprietà, in quanto l'evoluzione del sistema dipende dalle scelte (parliamo sempre di assegnazione degli operatori che è il *topic* di interesse) che vengono fatte. Abbiamo dato una rappresentazione del sistema, in modo tale da rendere il processo un MDP, quindi in modo tale che gli stati fossero tutti *Markov* e che fosse potenzialmente applicabile il RL per determinare le azioni da eseguire come scelta degli operatori in ciascuno stato.

Come abbiamo modellato il sistema e perché, per trovare gli stati, le azioni, le probabilità di transizione e tutto ciò di necessario per la definizione dell'MDP del sistema viene descritto nel paragrafo 6.1 che seguirà.

Partiamo dal dire che, per far sì che ogni stato sia Markov (ossia per esso valga la proprietà di Markov), la definizione dello stato dovrà essere ampia, ossia contenere un gran numero di informazioni in modo tale che ogni stato sia sufficientemente differenziato dall'altro e che possa contenere le informazioni sulla storia (ovvero il percorso che lo ha portato fino a lì) in modo che sia univocamente destinata la sua destinazione (eventualmente probabilistica), a seconda dell'azione che viene intrapresa.

Questa impostazione fa irrimediabilmente sì che il numero di stati che risulteranno sarà ampio. Per questo motivo, scegliamo di analizzare una configurazione del sistema molto semplice, la più semplice possibile, -quella con solo 3 operatori (e 3 WIP consentiti). Un alto numero di operatori fa anch'esso aumentare il numero degli stati, in quanto aumenta il numero delle combinazioni di operatori da assegnare.

Ciò che aspiriamo ad utilizzare, e al cui ottenimento sarà dunque finalizzato il codice è la funzione predefinita di Matlab `createMDP`, che crea il MDP sulla base dei dati inseriti: numero di stati, numero di azioni, stati terminali, probabilità di transizione e *Reward* per ciascuno stato dell'MDP. Successivamente, sempre mediante funzioni Matlab predefinite, -tutte contenute nell'apposito tool *Reinforcement Learning Toolbox-,* si crea, per il modello di processo specificato, il relativo ambiente (*environment*) per il RL, si

impostano i parametri per il learning, e poi si effettua il *training* (addestramento dell'agente per il RL) e la validazione dei risultati ottenuti.[18]

6.1 Modello di learning

Il riferimento al modello è l'allegato 2, il quale andiamo a spiegare nei suoi punti principali di funzionamento e nelle sue differenze (che sono sostanziali) rispetto al modello del sistema visto nel capitolo precedente, in modo simile a quanto fatto nel capitolo 5.

Per quanto riguarda la definizione delle variabili, le principali innovazioni e differenze rispetto al codice del processo sono che:

- non essendoci ottimizzazione interna al codice per l'assegnazione degli operatori, al contrario del caso precedente, tutte le variabili che tracciavano gli operatori (*W,Wt,Wp, assegnato*,…) non le troviamo più: l'assegnazione degli operatori è assegnata al meccanismo di RL;
- sono state eliminate le variabili *incorso* e *finisce*, che servivano per monitorare il processo, perché tutte le informazioni utili a tal fine saranno contenute nella nuova variabile *stratoinfo*, di tipo 'cell', che conterrà per ciascuno *strato* tutte le informazioni rilevanti;
- la definizione della matrice che memorizza gli stati, *S*, è stata modificata in modo che fosse adatta al nuovo contesto, e quindi che rappresentasse effettivamente degli stati Markov, come spiegato sopra. La matrice *S* adesso contiene:
    - i valori 1-33 rappresentano sempre i 33 task, ma qui il valore assegnato non è più il codice dell'operatore, bensì viene incrementato di 1 il valore della relativa casella, ogni volta che una lavorazione del tipo specificato (1-33) è terminata; le colonne 1-33 avranno avranno dunque un andamento progressivo con l'avanzare dello strato;
    - i valori 34-37 rappresentano i 4 operatori (3 operatori standard e 1 saldatore in questa configurazione, da modificarsi in caso di aggiunta di operatori standard),- in ordine-, quindi: la posizione 34 è relativa all'operatore standard '1', …, la posizione 37 è relativa al saldatore.

      Questi valori indicano, per ciascun operatore, la relativa posizione occupata (inteso come posizione fisica: in che stazione si trova l'operatore) nel caso in cui l'operatore sia disponibile (quindi non impiegato in alcuna lavorazione), altrimenti con quale job è impiegato nello stato in considerazione l'operatore, preceduto da un '-' in modo da differenziare le due situazioni.

      Quindi: nello stato di partenza '1', visto che gli operatori standard si trovano tutti nella posizione del dispatcher, le caselle 34-36 saranno tutte impostate sul valore '14'; ad esempio, nello stato 2 l'operatore 1 si trova occupato ad effettuare il task di trasporto numero 16, per cui la posizione 35 (*S(2,35)*) è pari a -16, la posizione 34 invece, contiene il valore 7, questo perché all'operatore 1 era

[18] https://it.mathworks.com/help/reinforcement-learning/ug/train-reinforcement-learning-agent-in-mdp-environment.html

stato assegnato il task 22, che finisce nella stazione con codice Matlab 7, che nello stato 2 è appena terminato. La posizione della matrice dedicata al saldatore, -cioè la 37-, poiché come sappiamo egli non si sposta dalla stazione di saldatura, che ha codice Matlab 6, sarà o 6 oppure 8,9,10, o 11, - che sappiamo essere i codici dei 4 task di saldatura del processo, preceduti da un '-'.

- I valori 38-45 (ci sono delle caselle extra, come nel codice dell'All.1, che resteranno vuote) sono i relativi valori della variabile *assegnatoconprio*, che qui memorizza per ciascuno strato (non 'stato') i task che risultano da assegnare agli operatori. Questi valori sono ritrovabili anche nella seconda colonna di *stratoinfo*.
- Le ultime due colonne, analogamente al codice del processo, riportano il numero di *items* completati in quello stato (valore 46) e se il processo è all'ultimo strato, quindi terminerà il quello stato (valore 47, corrispondente alla condizione END==1).

S viene in questo codice inizializzata con la condizione iniziale del processo, per quanto riguarda le posizioni degli operatori, quindi le posizioni 34-37 della matrice S alla riga di inizio (1) vengono impostate a 14, 14, 14, 6.

- La variabile *i* in questo codice indica lo strato, mentre in quello dell'All.1 indicava lo stato;

- sono state introdotte le variabili:

- *prod*, matrice con tante colonne quanti sono i task, quindi 33, che monitora la produzione, step by step, per ciascuno strato. Questa matrice sarà un estratto della più grande matrice precedentemente detta, *S*, ma sarebbe impossibile determinare i valori che andremo ad inserire in *prod*, a posteriori da *S*.
- *slast*, che viene aggiornata con l'ultimo stato definito. Gli stati saranno univoci, per cui questa variabile viene incrementata solo quando effettivamente si tratta di un nuovo stato, dopo apposita verifica di non preventiva presenza nella matrice S (tramite la funzione Matlab *ismember*, con l'opzione 'rows').
- *strato*, matrice a due colonne, che segna l'inizio e la fine di ciascuno strato di stati; il primo strato è costituito solo dallo stato iniziale '1', per cui viene inizialmente impostato ciò.
- *stratoinfo,* variabile fondamentale perché conterrà tutte le informazioni più dirette e rilevanti del processo. Ogni riga rappresenta uno strato, e per esso questa variabile, che come anticipato è di tipo cell ed ha 5 colonne, riporta:
  - in colonna 1 quali job finiscono in quello strato: la posizione 1 della riga 1 resterà vuota, perché essendo lo strato inziale, in esso non vi sono conclusioni di task precedentemente assegnati;
  - in colonna 2, quali task vengono assegnati in quello strato (analogia con la matrice *assegnatoconprio*); la posizione può eventualmente, ovviamente, rimanere vuota;
  - in colonna 3, i task che risultano in corso (quindi precedentemente assegnati); nello strato 1 (riga 1 dunque) anche questo valore rimarrà nullo, non potendoci essere task precedentemente assegnati e non ancora terminati, essendo lo strato-stato iniziale;

  - in colonna 4 i tempi (minimo e massimo) di lavorazione dei task assegnati e in corso, in questo ordine e tenendo l'ordine della priorità data al momento dell'assegnazione;
  - in colonna 5 il minimo dei range temporali della colonna 4: questo valore serve per determinare quale/i task finisce/scono nello strato successivo.
- *f*, variabile di riferimento per il task che termina in dato strato;
- *sald,* che indica per ciascuno strato l'occupazione o meno del saldatore;
- *maxrewardcum,* variabile numerica che parte da 0 e viene incrementata ad ogni passaggio di strato del valore massimo (ideale) del reward che è (sempre idealmente, -sottolineiamo-, perché nella pratica non è possibile ottenere il massimo reward in ogni strato) possibile ottenere in dato (nuovo) strato. Il valore di *maxrewardcum* che otteniamo a fine simulazione, -fatta sul tempo che vogliamo simulare-, è il valore che andremo ad inserire nelle opzioni del training (`trainOpts.StopTrainingValue`) come riferimento per il termine del training. Nella pratica, il training si fermerà una volta raggiunto il numero massimo di episodi, in quanto l'addestramento dell'agente non potrà mai toccare quel valore ideale impostato: è ciò a cui si vuole tendere. Il numero massimo di episodi viene anch'esso impostato nelle opzioni del training (`trainOpts.MaxEpisodes`), insieme agli altri parametri di `rlTrainingOptions`[19].
- *rcorr,* che per ogni strato memorizza il reward massimo che è possibile idealmente ottenere;
- *t,* che memorizza il tempo atteso di ciascuno strato, quindi riporta anche l'avanzamento temporale.

Dopo aver definito le variabili, e preventivamente al macro-ciclo che effettuerà tutte le operazioni che ci servono per ottenere i dati per il meccanismo di RL, ci preoccupiamo di aprire un file *.m,* nel quale andremo a scrivere ciò che ci serve per la definizione dell'MDP man mano che i valori vengono trovati. Ciò che serve per la definizione dell'MDP sono tutte le probabilità di transizione e relativi reward degli stati; il codice scrive queste informazioni, nel modo adatto, man mano che le trova. Dunque, apriamo un file vuoto 'write.m' in scrittura; il comando è il seguente: `fileID = fopen('write.m','w');` .

6.1.1 La parte hard del codice

Il modello del Il codice è, anche qui, racchiuso in un macro-ciclo che va avanti sinché la variabile END, nulla, non diventa 1. A partire dal secondo giro di questo ciclo, quindi dallo strato 2, si incomincia a scrivere sul file, che è lo scopo principale di questo codice/procedura. Seppur si parta dal secondo giro a scrivere, tutti gli strati devono e saranno rappresentati nel file. Ciò che viene scritto, inizialmente ed ad ogni cambio giro/strato, mediante l'istruzione: `fprintf(fileID,'%%stati strato %d\n\n',i-1);` , che come vediamo ha '*i-1*', quindi lo strato precedente, come riferimento. Ciò che viene scritto al giro 2 con questa istruzione è: `%stati strato 1` e a capo. Ad ogni nuovo strato, questa frase precederà nel file le istruzioni dell'MDP di tutti gli stati di quello strato (alla fine della spiegazione del codice mostreremo la parte inziale dello script che ne risulta, perché sia chiaro il risultato e ciò che serviva ottenere) .

[19] https://it.mathworks.com/help/reinforcement-learning/ref/rltrainingoptions.html

Il valore della variabile *prod* nello strato successivo, viene copiata al pari di come è nello strato attuale (di modo che non si perda l'informazione sui task già completati quando si procede nel codice).

Dopo di ciò inizia una sezione analoga a quella del codice del sistema visto in precedenza, nella quale si ricercano i task da assegnare, e che ha tanti cicli *while* quanti sono i job ammessi all'interno del sistema, quindi 4 in questo caso. Sempre analogamente, si definisce *daassegnareconprio* e le variabili di controllo dei risultati *nopdisp* e *sald*, che indicheranno per ciascuno strato, rispettivamente il numero di operatori disponibili e la disponibilità /non disponibilità del saldatore.

Successivamente, -e questa parte è diversa rispetto all'altro codice-, si verifica, un task alla volta per quelli che risultano da assegnare, se vi siano abbastanza operatori disponibili, o se il saldatore sia disponibile se si tratta di un task di saldatura, e in caso affermativo si inserisce il task in *assegnatoconprio*, variabile analoga a quella del codice precedente, e si effettuano le necessarie variabili di controllo e la relativa matrice che traccia il completamento.

Segue ancora una parte già presente nel primo codice, nella quale si memorizza il numero di task assegnati nello strato corrente (variabile *x*), si eliminano eventuali zeri da *assegnatoconprio,* si annullano i corrispondenti valori delle matrici che tracciano l'assegnamento e il completamento in caso di task di trasporto preliminari dei due percorsi di lavorazione gemelli, si riportano allo strato successivo gli attuali valori delle stazioni (variabile *St*) e della variabile *s*, si annullano eventualmente le variabili di percorso, si imposta eventualmente la variabile *Stt* su un valore diverso da 1 ecc.

Finita la parte precedente, alla riga 899 vediamo che viene trovato il minimo tra i range temporali dei task assegnati/in corso, che si trovano nella cella corrispondente alla quarta colonna dello strato corrente in *stratoinfo*; il task/i task corrispondente/i a tale tempo minimo è/sono quello/i che nello strato successivo termineranno. I valori che determinano tale range minimo, identificati nella variabile *c*, vengono riportati nella colonna 5 della variabile *cell stratoinfo,* e il task/i task che finisce/ono riportati nella colonna 1 della riga di *stratoinfo* successiva. Tutti i task assegnati o in corso nello strato corrente, che non siano tra quelli che finiscono nello strato successivo, vengono riportati nell'apposita colonna 3 di *stratoinfo*, nella riga successiva.

Nella variabile *f* vengono inseriti i task che finiscono, cioè, come detto, quelli che si trovano in `stratoinfo{i+1,1}`, per i quali tutti si procede ad aggiornare la relativa posizione nella matrice *prod*, riga *i+1,* e poi si procede con le medesime operazioni viste per i task che terminano nel codice precedente (righe 916-1043).

Da qui in poi vi è la parte fondamentale del codice, e fondamentalmente diversa concettualmente rispetto al codice di sistema. A partire dallo strato 2 (come spiegato prima, si scrive su file e si fanno i calcoli relativi ad uno strato in quello successivo, per questo motivo il primo viene saltato, o meglio, è solo propedeutico e viene analizzato quando i è pari a 2), per ciascuno stato dello strato, quindi si scansiona tutto lo strato mediante ciclo for (riga 1049), a cui attribuiamo l'indice *jj*.

Per ciascuno stato:

- se `enne(i-1)>0` , ossia ci sono operatori standard da assegnare (eventualmente assieme al saldatore):

vado a memorizzare nella variabile temporanea vettore *v* gli operatori disponibili: `v=find(S(jj,34:36)>0);` dopo di che calcolo la matrice delle combinazioni degli operatori disponibili, quindi delle azioni che è possibile effettuare nello specifico caso: `NPK=npermutek(v,enne(i-1));` ed effettuo la stessa procedura di definizione della matrice delle azioni *azioni* e di aggiornamento della matrice delle azioni per ogni stato *A* che abbiamo già visto nel precedente codice di sistema, con la differenza che qui non modellizzo l'assegnazione eventuale del saldatore, in quanto ininfluente per la procedura di RL.

- se `enne(i-1)` non è maggiore di zero, ossia non ci sono azioni rilevanti da intraprendere, nel senso che o non vi è nessun operatore da assegnare, o solo il saldatore:

NPK è pari semplicemente al vettore [0,0,0], che viene aggiunto in *azioni* solo nel caso in cui non sia ancora stato inserito, e viene indicata come unica posizione nella matrice *A*, posizione dello stato in questione.

- Per ciascuna azione che è possibile intraprendere nello stato in questione, - per cui si fa un ciclo *for* che va da 1 al numero di elementi nella matrice *A*, riga corrispondente allo stato corrente (riga 1086, `for w=1:numel(A{jj})`) -, si memorizza in *a* la combinazione che costituisce l'azione in analisi e, nello stato successivo, identificato dalla variabile *slast* :

  - si trascrive nelle caselle relative agli operatori standard (34-36) della matrice degli stati *S* i valori di tali caselle nello stato in questione (*jj*);

  - si aggiornano i valori 34-37 della matrice *S*, riga *slast*, con il numero del task che viene eventualmente assegnato a ciascun operatore standard secondo *a* o al saldatore se si tratta di un task di saldatura, preceduto dal segno meno, secondo la logica precedentemente descritta. I task da assegnare vengono identificati tramite uno scan sui valori successivi al 37 della riga jj-esima di S, dedicati a memorizzare i task da assegnare in ciascuno stato. Questa procedura include, nella sua scan, la diversificazione dei task normali da quelli di saldatura in modo che questi ultimi vengano assegnati al saldatore, quindi posti in posizione 37.

  - Si identifica il task che tra quelli assegnati termina nello stato *jj* in questione, e la posizione da lui occupata viene modificata nella sua stazione di arrivo o di lavorazione, in modo tale da indicare la disponibilità dell'operatore a cui il task è stato assegnato e la conclusione del task, sempre come da procedura precedentemente descritta.

  - Si trascrive il valore della variabile *prod* nello strato corrente, nelle caselle dedicate (le prime 33) di *S*, riga corrente *slast*, ed in caso di produzione maggiore di 0, si aggiorna anche la colonna 46-esima con il valore corrente della produzione totale.

  - In caso vi sia qualche task da assegnare in questo stato/strato (`if any(stratoinfo{i,2})`) , i codici dei task da assegnare vengono trascritti nelle apposite colonne, dalla 38 in poi, a seconda di quanti task risultino da assegnare, e occupando tante posizioni quanti operatori richiede il task (quindi 1 o 2).

- A questo punto si è completata la scrittura della riga *slast* della matrice *S* e si procede a verificare se sia già presente nella matrice, quindi sia un doppione e in caso negativo la variabile *s1* viene posta pari ad *slast* ed *slast* incrementato di 1, altrimenti *s1* viene posta pari all'indice della riga di cui la nuova riga *slast* è doppione.

- Adesso, si può scrivere sul file la transizione che è stata identificata, ossia a quale stato si arriva, con probabilità certa, se si sceglie l'azione in questione, nello stato in analisi. In codice: `fprintf(fileID,'MDP.T(%d,%d,%d) = 1;\n',jj,s1,A{jj}(w));` ; questa istruzione genera un testo del tipo: MDP.T(1,2,1) = 1; che, considerato questo esempio, sta a significare che dallo stato 1, se effettuo la scelta dell'azione 1 arrivo con probabilità certa (pari a 1) allo stato 2.

- Utilizzando la variabile ausiliaria *somma*, che viene inizializzata a 0 ad ogni giro del ciclo for sulle azioni, quindi sempre prima delle istruzioni che stiamo per spiegare, viene calcolata la sommatoria delle distanze che sono da percorrere in base all'assegnazione degli operatori in questione. Una volta calcolata questa sommatoria, si trova il reward (o ricompensa) che dà la scelta dell'azione *A{jj}(w)* , identificata con *a*-, nello stato in questione *jj*.

La ricompensa è calcolata come una penalità, - individuabile mediante segno meno-, per ogni chilometro percorso, ed una ricompensa effettiva, quindi un valore positivo, pari a 200, in ogni caso in cui viene terminato un job. Questo implica che se gli operatori assegnati non devono spostarsi, non viene data alcuna penalità (reward pari a 0), stessa cosa se non viene terminato il job in seguito alle azioni intraprese. In particolare, e per chiarire meglio come sono stati impostati i reward per il meccanismo di RL:

`R=round(-(somma/10),4);` : il reward della situazione in analisi *R* viene impostato pari alla somma delle distanze che è necessario percorrere per effettuare la combinazione di azioni scelta, diviso 10, arrotondato alla 4^ cifra decimale mediante la funzione round di Matlab.

```
if R==-0
   R=0;
```

`end` : in caso venga fuori '0', ossia non vi sono distanze da percorrere per gli operatori, '-0' viene trasformato in '0' per pure questioni di calcolo. Adesso:

```
if produzione>0 && out(produzione)==i
   R=R+200;
```

`end` , che significa che nel caso in cui nello strato corrente *i* vi sia un incremento della produzione (quindi un job ha terminato tutte le lavorazioni ed esce dal sistema) viene assegnato un reward positivo scelto pari a 200 che viene sommato al Reward precedentemente calcolato.

- A questo punto, una volta che abbiamo calcolato il Reward della situazione corrente, possiamo stampare su file la seconda riga necessaria per ciascuno stato corrente e azione: quella relativa alla ricompensa che si ottiene passando dallo stato attuale *jj* a quello individuato *s1*.

L'istruzione relativa alla stampa su file della ricompensa è: `fprintf(fileID,'MDP.R(%d,%d,%d) = %g;\n',jj,s1,A{jj}(w),R);` ; facendo anche in questo caso un esempio, -estrapolato dallo script

finale-, MDP.R(1,2,1) = -13.6399; significa che la ricompensa che si ottiene passando dallo stato 1 allo stato 2 mediante l'azione 1 (stessa casistica di sopra) è una penalità di 13.6399.

- A questo punto resta da fare l'ultima cosa prima che il ciclo sulle azioni si ripeta fino a concludersi, e poi quello sugli stati dello strato precedente a quello corrente si ripeta fino a concludersi, ed è il calcolo della variabile *rcorr*, nella quale si va a sommare il valore della ricompensa calcolata, se si è al primo giro o se essa è maggiore del valore precedente. Questa variabile, come preventivamente esplicato, memorizza per ogni strato il valore massimo di ricompensa idealmente ottenibile, e contribuisce alla somma totale cumulata delle ricompense massime ottenibili, che verrà aggiornata nel resto del codice.

Concluso il ciclo che analizza tutti gli stati dello strato precedente a quello corrente, e ne scrive su file i parametri in vista della procedura di RL:

- viene incrementato di 1 il valore della variabile *jj*;
- si aggiorna il valore del massimo reward cumulato *maxrewardcum* che è possibile ottenere fino allo strato precedente allo strato corrente *i*: `maxrewardcum=maxrewardcum+rcorr(i-1);`
- si riporta a zero l'intera riga della matrice *S* di posizione *slast*+1, che potrebbe contenere dei valori, e in quel caso vanno cancellati;
- si fa un check per verificare che non ci siano stati errori nell'aggiornamento della matrice *S*, per cui si cercano le righe univoche di *S*, mediante la funzione *unique,* con opzioni *'stable'* e '*rows*' e si mette il risultato dentro la variabile *ia*; se il valore *ia(slast*) non coincide con *slast*, viene visualizzato un messaggio di errore. [Il codice è stato eseguito con la configurazione detta e non vengono evidenziati errori]
- vengono memorizzati i valori dello strato corrente: l'inizio dello strato i-esimo viene posto pari a *jj*, e la fine ad *slast*.

Se *i* è pari ad 1 la parte precedente non è stata eseguita (tutta la parte che va da quando abbiamo detto "a partire dallo strato 2" sino a qui) e per lo strato 1, che coincide, -ribadiamo-, con lo stato 1 unicamente, tutto ciò che vi è da fare viene fatto ed è memorizzare nelle posizioni dalla 38 e successive della riga 1 di *S* il valore i-esimo di *assegnatoconprio*, ossia i task da assegnare nello strato/stato 1.

L'ultima operazione da fare è aggiornare il valore del tempo nello strato corrente *i*; esso viene posto uguale al tempo dello strato precedente *i-1*+ il valore intermedio del range identificato in *stratoinfo*, posizione 5: `tempo(i)=tempo(i-1)+(stratoinfo{i,5}(1)+stratoinfo{i,5}(2))/2;`

Le ultime righe sono dedicate al check sul tempo, che se superiore a quello desiderato per la fine della ricerca, fa sì che venga posta la variabile END pari a 1 in modo che il codice si concluda a quell'i e la posizione 47-esima dello strato precedente *i-1* posta ad 1 ad indicare che è l'ultimo.

Prima che il macro-ciclo ricominci, come è necessario, la variabile *i* viene incrementata di 1.

6.2 Procedura di RL

Facendo girare il codice, otteniamo uno script, e dei dati che salviamo. Andiamo a verificare quale sia l'ultimo strato la cui fine rientra nelle 7 giornate da simulare, ossia entro un tempo cumulato pari a 201600 s, e andiamo ad eliminare dallo script ottenuto le righe di testo relative a stati/strati che non consideriamo.

Andiamo a verificare il numero di stati e di azioni complessivamente ottenuti dalla procedura Matlab, e possiamo andare a definire l'MDP, aggiungendo in cima allo script ottenuto le istruzioni:

```
slast=48478;
MDP=createMDP(slast,16); .
```

Abbiamo dunque necessità di costruire un MDP con 48478 stati e 16 azioni, per la situazione corrente.

La costruzione dello script non è ancora terminata in quanto mancano le righe relative alle azioni cosiddette "vietate", ossia quelle azioni che rientrano nella matrice delle azioni, quindi possono essere eseguite in almeno uno stato, ma che in uno specifico stato non è possibile eseguire, risultando quindi "vietate" nello specifico stato. Poiché nella costruzione dello script mediante il codice precedentemente descritto, sono state prese in considerazione, per ciascuno stato, solamente le azioni effettivamente eseguibili, bisogna specificare cosa accade, per ciascuno stato, in corrispondenza di quelle azioni. Bisogna aggiungere queste informazioni affinchè l'MDP sia completo: in caso contrario, la funzione predefinita di creazione dell'MDP non è applicabile. Questa funzione infatti, si basa sulla creazione di matrici con dimensioni basate sul numero di azioni e stati fornito, il cui riempimento dei valori ha necessità delle informazioni (probabilità di transizione e reward) di tutti gli stati con tutte le azioni.

Per modellare nell'MDP il fatto che certe azioni possono essere vietate in certi stati, si indica come probabilità di transizione la probabilità certa di rimanere nello stesso stato, e come reward abbiamo scelto una penalità di 40, in modo che fosse più alta delle penalità attribuite in caso di azioni non vietate.

Vi è dunque una parte aggiuntiva, che riportiamo, che abbiamo eseguito per la scrittura delle righe aggiuntive, che vanno aggiunte nello script ottenuto:

```
%aggiunta azioni vietate nell'MDP
fprintf(fileID,'%%Azioni Vietate:\n\n');

for j=48431:strato(1450,2)
    for w=1:16
        if ~ismember(w,A{j})
            fprintf(fileID,'MDP.T(%d,%d,%d) = 1;\n',j,j,w);
            fprintf(fileID,'MDP.R(%d,%d,%d) = -40;\n',j,j,w);
        end
    end
end
```

L'ultima istruzione da aggiungere per il completamento dell'MDP è l'indicazione di quali siano gli stati terminali, mediante l'istruzione:

`MDP.TerminalStates = ["s2";"s1];` dove tra virgolette vengono inseriti, separati da un punto e virgola, i nomi ("s" seguita dal numero dello stato terminale) degli stati terminali, che sono tutti quelli dell'ultimo strato (qui "s1" ed "s2" sono stati inseriti solo come esempi, perché gli stati terminali sono tanti).

A questo punto, si crea l'ambiente MDP per il Reinforcement Learning per il nostro modello di processo, mediante l'istruzione: `env = rlMDPEnv(MDP);` , si specifica una funzione di reset che riporta all'agente allo stato inziale, che è sempre il primo: `env.ResetFcn = @() 1;` e si fissa il generatore casuale per la riproducibilità: `rng(0)`. La funzione di reset è richiamata all'inizio di ogni episodio di training e di simulazione.

Lo step successivo è creare un agente Q-learning: si crea innanzitutto una Q tabella usando le azioni e osservazioni specificate dall'ambiente MDP; impostiamo il tasso di apprendimento della rappresentazione a 1. I comandi sono:

```
obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env);
qTable = rlTable(obsInfo, actInfo);
qRepresentation = rlQValueRepresentation(qTable, obsInfo, actInfo);
qRepresentation.Options.LearnRate = 1;
```

Adesso, si crea effettivamente l'agente Q-learning usando la precedente rappresentazione tabellare, e configurando l'esplorazione *Epsilon-Greedy* (di Q-learning, esplorazione, e tutti i concetti di RL abbiamo parlato nel capitolo 2, dedicato alla teoria del RL):

```
agentOpts = rlQAgentOptions;
agentOpts.DiscountFactor = 1;
agentOpts.EpsilonGreedyExploration.Epsilon = 0.9;
agentOpts.EpsilonGreedyExploration.EpsilonDecay = 0.01;
qAgent = rlQAgent(qRepresentation,agentOpts);
```

La procedura che segue è fondamentale: si tratta dell'addestramento (*training*) dell'agente. Prima, si devono impostare i parametri desiderati per il training; si definisce: `trainOpts = rlTriningOptions;` e successivamente ne si compilano i parametri:

- il numero massimo di step per episodio, lo si può impostare di poco superiore al numero degli strati, che corrispondono agli step degli episodi;
- il numero massimo di episodi, più è alto e più sarà accurato il training, ma anche più lungo, quindi conviene impostarlo facendo dei tentativi. Bisogna considerare che il training effettuerà tutti gli episodi: per quanto spiegato in precedenza, non è possibile che il nostro training si fermi perché raggiunge la condizione di stop sul training value (che vediamo a breve), per cui si fermerà quando avrà effettutato il massimo numero degli episodi previsti.
- il criterio per terminare il training, possiamo impostare "EpisodeReward", ossia l'ottenimento di un dato reward prestabilito in un qualsiasi episodio, o "AverageReward", ossia il raggiungimento di una determinata media dei reward degli episodi.

- StopTrainingValue, ossia il valore a cui deve far riferimento la condizione di terminazione (punto precedente);
- e la "ScoreAveragingWindowLength", lunghezza della finestra per la media, impostazione della lunghezza della finestra su cui calcolare eventualmente la media dei reward.

Si effettua il training; il comando è: (ad esempio scegliamo il nome) `trainingStats=train(qAgent,env,trainOpts).` Questo comando fa partire il training dell'agente per il RL, il quale può richiedere anche molto tempo per essere completato in caso di un MDP grande.

Una volta terminato il training bisogna validare i risultati della procedura, e lo si fa simulando l'agente nell'ambiente di addestramento usando la funzione *sim* .

`Data = sim(qAgent,env);` salva i dati della simulazione per la validazione nella variabile Data, che sarà di tipo *Timespace* . Accedendo a *Data,* -al massimo effettuando qualche interrogazione-, è possibile visualizzare azioni, reward, e tutti i risultati della simulazione.

Per l'impostazione data alla procedura, sia a livello di codice che di impostazione delle ricompense, essa dovrebbe restituire dei risultati simili a quelli che si otterrebbero attraverso le simulazioni fatte in modo più classico, come visto nel capitolo precedente.

Vi è una parte, relativa alla procedura di RL applicata al nostro sistema, che può risultare problematica: il fatto che gli stati del sistema non siano naturalmente degli stati Markov fa sì che, come abbiamo visto, dobbiamo esprimerli in modo che essi lo diventino. Questo fa sì che il numero di stati necessari per costruire l'MDP sia alto, e ciò implica che la complessità computazionale aumenti, rendendo lunga e onerosa, ed eventualmente anche impossibile con la metodologia proposta, effettuare il learning e trovare una soluzione al problema.

Ciò che si può fare per ovviare a questo problema è: utilizzare una metodologia diversa, tenendo la stessa impostazione, usare un calcolatore con prestazioni elevate (in particolare di RAM), suddividere il problema in vari sotto-problemi, quindi risolvere degli MDP ridotti.

Considerazioni finali

Dall'analisi fatta sul sistema, anche mediante simulazione, è emerso che vi è una configurazione alternativa rispetto a quella attualmente impiegata dall'azienda, che risulterebbe per essa favorevole. L'azienda attualmente impiega 9 operatori standard ed un saldatore; lo studio ha evidenziato che la riduzione ad 8 del numero di operatori standard, se accompagnata da un numero massimo di job autorizzati entro il processo pari a 4, porta ad un aumento in termini di performance produttiva (Throughput) e utilizzo delle risorse in questione, - che comunque mantengono un buon margine di tempo di inutilizzo-, e ad una migliore distribuzione del lavoro tra gli operatori, ossia un minor sbilanciamento nel lavoro eseguito, degli operatori standard.

Dalla panoramica sul Reinforcement Learning si intuisce quanto sia promettente per l'ingegneria la cosiddetta "intelligenza artificiale", che può andare ad integrare, e in alcuni casi anche sostituire, i metodi di allocazione delle risorse classici, utilizzati anche nel caso di questo studio.

Emerge altresì dal lavoro di tesi, che nonostante l'incontestabile enorme potenzialità dei metodi di AI di learning, essi non sono adatti a tutte le situazioni o necessariamente migliori di quelli tradizionali, ma che molto della loro utilità dipende dal caso in esame. Se il caso in esame è complesso, -soprattutto se il problema non è esprimibile in modo semplice, con un ridotto numero di stati- , l'applicazione dei metodi di RL può risultare eccessivamente onerosa a livello computazionale e rendere il metodo in definitiva inadatto.

Sono numerosi i metodi per il RL, e se ne possono creare anche di nuovi, ma c'è da chiedersi se lo sforzo da effettuare, se non per questioni di ricerca ma per un'applicazione in campo pratico aziendale, sia calibrato poi dai risultati ottenuti e da ciò che il loro utilizzo determina. Probabilmente, questo sforzo va visto come un investimento per quelle aziende i cui processi sono tali, e le aziende sufficientemente grandi, da supporre un beneficio finale superiore agli sforzi effettuati. O ancora, risultano sicuramente positivi per quei singoli processi/problemi strutturati o strutturabili in modo tale da poter beneficiare dei vantaggi del learning senza dover effettuare grandi sforzi. Oppure ancora, per quei problemi/sistemi che traggono un grande beneficio dal solo fatto che il metodo utilizzato sia flessibile, senza alcuna difficoltà adattabile alla specifica, e sempre diversa, situazione.

Bibliografia:

1. Richard S. Sutton, Barto, Andrew G., *Reinforcement Learning: An Introduction*, MIT Press, 1998, ISBN 0-262-19398-1
2. Ronald A. Howard, *Dynamic Programming and Markov Processes*, The M.I.T. Press, 1960.
3. Olle Häggström (2002), *Finite Markov Chains and Algorithmic Applications*, Cambridge University Press, ISBN 0-521-81357-3
4. Rebeca Gomes de Hollanda Cavalcanti, *The study of workload balance and the influence of buffers size on a pallet factory using simulation*. Master's degree thesis, Politecnico di Torino, 2016.

Sitografia:

5. https://it.wikipedia.org/wiki/Apprendimento_per_rinforzo
6. https://it.wikipedia.org/wiki/Processo_decisionale_di_Markov
7. https://en.wikipedia.org/wiki/Markov_decision_process
8. https://it.wikipedia.org/wiki/Processo_markoviano
9. https://towardsdatascience.com/reinforcement-learning-demystified-36c39c11ec14
10. https://en.wikipedia.org/wiki/Reinforcement_learning