Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Modellazione di oggetti cognitivi per la gestione energetica efficiente e la qualità indoor in edifici intelligenti

Emilio Greco, Antonio Francesco Gentile

RT- ICAR-CS-20-03 **Giugno 2020**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 8-9C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sezione di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.icar.cnr.it*
– Sezione di Palermo, Via Ugo La Malfa, 153, 90146 Palermo, URL: *www.icar.cnr.it*

Indice generale

Introduzione

I problemi di schedulazione che generalmente vengono trattati in letteratura sono spesso statici, ovvero le attività sono note in anticipo e i vincoli sono fissi, tuttavia i risultati prodotti anche se soddisfacenti non tengono conto del fatto che ogni programma di schedulazione nella vita reale è soggetto a eventi inaspettati: il numero di attività ed i vincoli che descrivono il problema potrebbero cambiare anche in maniera sostanziale. In questi casi, è necessaria una nuova soluzione in un tempo preferibilmente breve. Inoltre, questa nuova soluzione deve essere non troppo lontana da quella precedente in modo da non stravolgere l'evoluzione del sistema.

Approcci alla soluzione di questi tipi di problemi li troviamo in letteratura annoverati sotto diversi nomi: Job shop scheduling[1], Resource Constrained Project Scheduling Problem (RCPSP)[2], Dynamic Scheduling[3].

Questi problemi di ottimizzazione richiedono la programmazione di una serie di attività tenendo conto dei vincoli temporali e delle risorse adoperate in un certo istante. Questo insieme di problemi ha una elevata complessità computazionale ed appartengono alla classe di problemi chiamati "NP-Hard[4]" [Blazewicz et al., 1983]. L'utilizzo di algoritmi euristici [5] convenzionali per risolvere questa tipologia di problemi richiede elevati tempi di esecuzione che non consentirebbero l'applicabilità in un sistema real-time. L'approccio che si vuole utilizzare quindi è quello di applicare gli algoritmi del Reinforcement Learning (RL) per risolvere questa classe di problemi tenendo conto dei tempi di esecuzione e delle risorse impiegate.

Nella prima parte dell'elaborato viene introdotto il concetto di "*apprendimento per rinforzo"* come metodologia per la risoluzione di problemi complessi. Dopo una breve descrizione degli algoritmi classici del RL e delle loro caratteristiche viene affrontata la parte progettuale del sistema.

Nella seconda parte dell'elaborato, vengono messe in evidenza le caratteristiche tecniche e strutturali del problema e le scelte progettuali adottate per risolverlo. Particolare enfasi viene data alle scelte che hanno portato alla trasformazione da un algoritmo classico di schedulazione ad un algoritmo di schedulazione dinamica, in grado di sfruttare l'auto-apprendimento come elemento di forza per adattarsi all'ambiente. Vedremo come l'algoritmo realizzato sarà in grado di rispondere a più richieste provenienti da utilizzatori differenti e quindi da ambienti diversi, ma anche come la variabilità dei parametri che definiscono lo stesso problema (tempo massimo di differimento, tempo di esecuzione, potenza assorbita, etc.) ha portato all'implementazione di un modello dinamico che di volta in volta si adatta per risolvere un problema specifico. La necessità di realizzare un algoritmo real-time e stand-alone ha spinto la progettazione verso un sistema che ottimizza il più possibile le risorse

[1] Scheduling su macchine parallele scorrelate, dove n lavori devono essere processati da m macchine diverse disposte in parallelo, dove l'obiettivo è assegnare i lavori alle macchine in modo tale da minimizzare il tempo totale di completamento.

[2] Consiste nel ricercare uno schedule ottimo per le attività di un progetto che soddisfi non solo i vincoli di precedenza, ma che provveda anche ad una corretta allocazione delle risorse disponibili quali: manodopera, materie prime, etc.

[3] La pianificazione dinamica è una tecnica utilizzata dai moderni calcolatori, in cui le istruzioni non vengono eseguite in base all'ordine di arrivo, ma piuttosto alla disponibilità delle risorse computazionali disponibili.

[4] Nella teoria della complessità computazionale, descrivono quei problemi non deterministici in cui non esiste un algoritmo che è in grado di risolvere velocemente (in tempo polinomiale) il problema.

[5] l'algoritmo euristico riesce a ricavare una soluzione approssimativamente vicina a quella ottima in problemi altrimenti irrisolvibili con la matematica classica.

computazionali e di memoria consentendone l'istallazione anche su dispositivi a basse prestazioni. La maggior parte dei dispositivi IoT che troviamo in commercio usano una tecnologia on-cloud per elaborare i dati acquisiti. Tecnica che da una parte consente di abbattere i costi di produzione e dall'altra la fidelizzazione gli utilizzatori. In questo progetto vogliamo svincolarci da tecnologie proprietarie e realizzare un prodotto plug and play, installabile su dispositivi a basso costo ed open source. La scelta progettuale che ha consentito il raggiungimento di questo obbiettivo è stata l'accurata definizione di una funzione filtro in grado di discriminare le azioni ammissibili dalle azioni non ammissibili a partire da un dato stato del sistema. Quest'approccio ha consentito una riduzione sostanziale dei tempi di elaborazione dell'algoritmo, sia perché ha ridotto il numero delle scelte che l'algoritmo deve operare in ogni fase di esecuzione, e sia perché si è ottenuto una riduzione dei dati da archiviare a seguito della riduzione degli stati da visitare. Questo ha consentito l'uso di strutture di archiviazione dati come gli alberi al posto delle matrici con sostanziali benefici di ingombro di memoria.

Nella terza parte dell'elaborato vengono fornite le nozioni di base per l'utilizzo dell'ambiente di sviluppo e di addestramento[6]. Viene descritto in particolare l'ambiente Openai Gym, uno dei più diffusi ambienti di addestramento e vengono presentati i primi risultati prodotti in ambiente simulato. Risultati ottenuti a partire da due algoritmi classici del RL : SARSA e Q-Learnig. I test sono stati effettuati sulla base di tre casi studio, utilizzando dati assegnati staticamente.

Il Reinforcement Learning

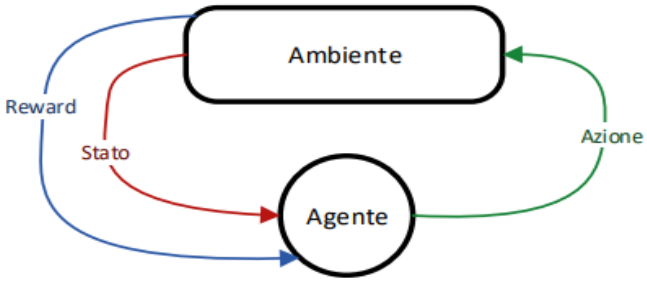
L'obiettivo della ricerca in machine learning è fornire ai calcolatori l'abilità di apprendere automaticamente un comportamento sulla base di informazioni ed esempi, senza essere programmati esplicitamente per svolgere un determinato compito. In quest'ambito i metodi di Reinforcement Learning cercano di determinare come un agente razionale debba scegliere determinate azioni da eseguire a partire dalla conoscenza dello stato corrente del sistema, perseguendo l'obiettivo di massimizzare una sorta di ricompensa totale per raggiungere uno stato terminale a lui noto. La ricompensa totale è determinata sulla base di una sequenza di reward (ricompense) che l'agente ottiene nell'eseguire le singole azioni che portano al raggiungimento dello stato terminale. Il meccanismo fondamentale di tutti gli algoritmi noti del Machine Learning è il miglioramento automatico del comportamento mediante l'esperienza accumulata in fase di addestramento. Ciò vuol dire che un programma di RL dovrà scoprire, mediante ripetute prove, quali azioni permetteranno di ottenere la ricompensa maggiore.

L'idea di base che sta nella formulazione di un problema di RL è catturare gli aspetti più importanti di un problema reale, facendo interagire un agente, che è in grado di apprendere il problema in modo da fargli raggiungere l'obiettivo prefissato. Per ottenere ciò è opportuno che l'agente sia in grado di operare con l'ambiente e osservarne lo stato ad ogni istante[7]. L'agente dovrà essere in grado di eseguire azioni le quali avranno effetto sull'ambiente modificandone lo stato. Inoltre all'agente dovrà

[6] Il processo attraverso cui i software di intelligenza artificiale sviluppano un certo grado di intelligenza viene chiamato addestramento.

[7] C'è una netta analogia con i sistemi di controllo retroazionati. Occorre individuare delle grandezze che governano il sistema, misurale, confrontarle con un valore obbiettivo e poi occorre intervenire con una attuazione per far evolvere il sistema.

essere assegnato un obiettivo da perseguire o più obiettivi se si vuole che il sistema evolva attraversando alcuni stati desiderati e ne eviti altri indesiderati. La formulazione del problema di learning si basa dunque su questi tre aspetti: osservazione, azione e obiettivo.



La continua iterazione tra agente ed ambiente permette una continua acquisizione di conoscenza da parte dell'agente. Conoscenza che deve sfruttare in modo da massimizzare la ricompensa finale. La conoscenza acquisita non deve essere però preponderante sulle scelte dell'agente che allo stesso tempo deve esplorare nuove soluzioni in modo da scegliere azioni migliori nelle esecuzioni future. Bisogna considerare che l'agente interagisce con un ambiente stocastico, questo comporta che ogni azione dovrà essere provata più volte per ottenere una stima reale della ricompensa prevista.

Elementi del Reinforcement Learning

Oltre all'agente e l'ambiente, è possibile identificare quattro principali sotto elementi che concorrono a definire un problema di Reinforcement Learning: la reward function, la value function, la policy e, eventualmente, un modello dell'ambiente.

Abbiamo già parlato della reward function come quella funzione che associa ad una coppia stato-azione un valore numerico. Tale valore identifica nell'immediato la bontà nell'intraprendere una azione in un certo momento o in un determinato stato.

Se consideriamo il fatto che qualsiasi decisione che si prende nella vita reale richiede una valutazione preventiva tra rischi e benefici, ci si rende immediatamente conto che un modello decisionale che tenga conto della reward function come unico elemento di valutazione non porti ai risultati desiderati. In altri termini non è sufficiente per produrre una soluzione al problema che tenga conto dei rischi e dei guadagni a lungo termine.

In questo caso è necessario definire una seconda funzione, detta funzione utilità, che associa ad ogni stato un valore numerico che rappresenta la ricompensa totale che l'agente si può aspettare di accumulare nel futuro, partendo da quello stato.

Tuttavia le decisioni potranno essere prese solo sulla base di "valori stimati" (metodo bootstrap), questo perché l'obiettivo dell'agente è raggiungere un nodo terminale e massimizzare la ricompensa totale ottenuta nel raggiungerlo. La ricompensa totale sarà nota all'algoritmo soltanto al

raggiungimento dell'obbiettivo. Sfortunatamente determinare i valori è più complicato che determinare i reward, questo perché i secondi vengono forniti all'agente direttamente dall'ambiente, mentre i primi devono essere stimati e ri-stimati mediante le sequenze di osservazioni dell'agente.

In problemi in cui è noto il quarto elemento che definisce un problema di Reinforcement Learning, cioè il "modello dell'ambiente", allora applicando una particolare funzione valore nota come equazione di Bellman

$$U(s) = R(s) + \gamma \cdot \max_{a} \sum_{s'} T(s,a,s') \cdot U(s')$$

sarà possibile ottenere una soluzione del problema risolvendo un sistema di N equazioni in N incognite.

In questo caso con il termine "modello dell'ambiente" si intende la matrice di transizione T(s,a,s') che descrive le probabilità di transizione ottenuta nel passaggio di stato tra lo stato s ed lo stato s' seguendo una certa azione a.

Se consideriamo il classico esempio del problema del lago ghiacciato (frozen lake), dove l'agente deve decidere il percorso migliore che porta da un punto iniziale (A2) ad uno finale (C2) evitando le buche (C3), se si conoscono le probabilità di transizione per ogni stato, allora è possibile giungere ad una soluzione in maniera molto semplice risolvendo l'equazione di Bellman.

Utilizzando un metodo di calcolo iterativo per la soluzione dell'equazione di Bellman, si dimostra la convergenza dell'algoritmo dopo tre iterazioni:

Il metodo risolutivo visto prende il nome di **programmazione dinamica**. È applicabile a problemi di cui è noto il modello dell'ambiente. Un limite all'applicabilità di questo approccio risolutivo

consiste nel fatto che ciascuno dei passaggi iterativi contiene una scansione completa dello spazio degli stati e se il numero degli stati è alto, i tempi di convergenza sono molto lunghi.
I metodi DP convergono alla soluzione ottima con un numero di operazioni polinomiali rispetto al numero di stati m e azioni n. I metodi DP aggiornano le stime dei valori degli stati, sulla base delle stime dei valori degli stati successori, ovvero aggiornano le stime sulla base di stime passate. Ciò rappresenta una proprietà speciale, che prende il nome di bootstrapping. Diversi metodi di RL eseguono bootstrapping, anche metodi che non richiedono un perfetto modello dell'ambiente, come richiesto dai metodi DP.
Un altro approccio per la soluzione di questi problemi, prende il nome di metodo **Monte Carlo.** Considerando il fatto che quasi sempre è complicato realizzare un modello di un ambiente per le dinamiche che vi sono dietro, allora si preferisce lasciare questo compito ad un algoritmo. Individuati un certo numero di stati esaustivi e mutuamente esclusivi che descrivono un ambiente ed un certo numero di azioni necessarie per descriverne l'evoluzione, il metodo Montecarlo di fatto mira alla ricerca di percorsi (soluzioni del problema) da un nodo (stato) iniziale verso i nodi terminali facendo delle stime di bontà dei percorsi ottenuti sulla base della somma totale di reward. Stima ottenuta in media negli episodi passati. Questo metodo si basa sul presupposto che l'esperienza sia divisa in episodi, e solo una volta portato a termine un episodio avviene la stima dei nuovi valori e la modifica della **policy** .
La policy in genere viene definita come una regola stocastica mediante la quale l'agente seleziona l'azione in funzione dello stato corrente. Dato che l'obiettivo dell'agente è massimizzare la quantità totale di reward ottenuta nel tempo, la policy dovrà essere greedy rispetto alla funzione valore, preferendo la scelta di azioni stimate come migliori dalla funzione valore, dove per migliore azione, si intende l'azione con valore maggiore in un dato stato.
A differenza dei metodi DP che calcolano i valori per ogni stato, i metodi Monte Carlo calcolano i valori per ogni coppia stato-azione, questo perché in assenza di un modello, i soli valori di stato non sono sufficienti per decidere quale azione è meglio eseguire in un determinato stato.
Inoltre i metodi Monte Carlo differiscono dai metodi DP per due motivi. Per prima cosa i metodi MC apprendono direttamente da campioni di esperienza, e quindi è possibile apprendere le dinamiche dell'ambiente in linea con l'acquisizione dell'esperienza da parte dell'agente in assenza di un modello.
L'apprendimento mediante **Temporal Difference (TD)** è una combinazione delle due idee. Si apprende direttamente dall'esperienza in assenza di un modello come i metodi MC e come i metodi DP, i metodi TD aggiornano le proprie stime basando il calcolo in parte su stime passate (eseguono bootstrap). Ovvero ogni scelta che viene fatta, viene "ponderata" in funzione dell'esperienza acquisita dalle precedenti iterazioni.

L'agente deve sfruttare quello che già conosce in modo da massimizzare la ricompensa finale, ma nel contempo deve esplorare in modo da scegliere azioni migliori nelle esecuzioni future.

L'agente in fase di apprendimento basa le proprie decisioni sullo stato percepito dell'ambiente, costruendo di volta in volta "il modello dell'ambiente". Per modello si intende un'entità in grado di simulare il comportamento della parte di realtà rappresentata. Per esempio, dato uno stato e un'azione, il modello è in grado di predire il risultato del prossimo stato e prossimo reward.

Affinché l'algoritmo sia in grado di costruire un modello dell'ambiente che ne descrive in tutto la sua dinamica è indispensabile che il problema così come formulato, goda della cosiddetta **proprietà di Markov.** Ovvero ogni stato che rappresenta l'ambiente non contiene memoria e quindi è possibile conoscere o determinare lo stato futuro in cui evolve il sistema a seguito di una azione basandoci solo sulle informazioni contenute nella rappresentazione dello stato attuale.

Verificato che un problema gode della proprietà di Markov, allora sarà possibile costruire ed utilizzare un modello del problema realizzato attraverso una matrice di conoscenza Q che consideri tutte le coppie stato-azione del sistema.

Algoritmo Q_Learning

Uno degli algoritmi più importanti relativi alla risoluzione di problemi di apprendimento è il Q-Learning e si basa sull'utilizzo della funzione Q(s,a). Apprendere la funzione Q equivale ad apprendere la politica ottima. Per apprendere Q, occorre un modo per stimare i valori di addestramento per Q a partire solo dalla sequenza di ricompense immediate r in un lasso di tempo. La funzione Q viene aggiornata utilizzando la seguente formula:

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

L'implementazione degli algoritmi appena trattati avviene attraverso l'utilizzo di una matrice avente un numero di righe pari al numero degli stati e un numero di colonne pari al numero delle possibili azioni. In ogni cella viene memorizzata una stima della funzione stato-azione Q(s,a).

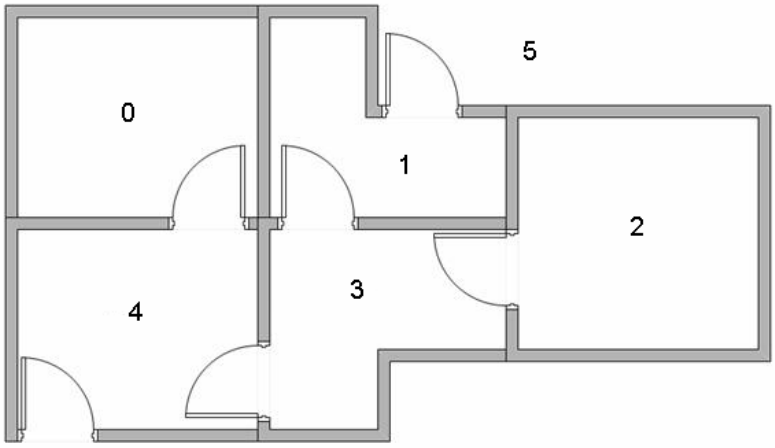
Durante la risoluzione di un problema di Reinforcement Learning bisogna essere in grado di bilanciare la fase di esplorazione e di sfruttamento. L'esplorazione (exploration) dello spazio delle azioni permette di scoprire azioni che portano a ricompense migliori. Un agente che esplora solamente difficilmente riuscirà a convergere ad una policy ottima. Le azioni migliori vengono scelte ripetutamente (sfruttamento) perché garantiscono una ricompensa massima (reward). Il bilanciamento può essere ottenuto attraverso la tecnica ε-greedy. Questo metodo permette di scegliere l'opzione greedy per la maggior parte delle volte al fine di sfruttare l'informazione immagazzinata fino a quel momento e massimizzare la ricompensa totale e con probabilità ε di scegliere una azione in maniera casuale così da poter esplorare eventuali policy maggiormente produttive. Valori di ε troppo grandi portano a tempi di convergenza elevati mentre valori troppo piccoli non permettono di trovare la policy ottima. Il parametro α $\in [0,1]$ influenza notev
gli algoritmi infatti per valori di α molto piccoli l'apprendimento è rallentato, mentre per valori di alpha troppo elevati l'algoritmo rischia di non convergere. Nell'implementazione degli algoritmi si inizia con un valore molto grande per poi farlo decrescere all'aumentare delle esplorazioni. Le prestazioni degli algoritmi presentati sono influenzati anche dalla inizializzazione della matrice Q(s,a) e dalla scelta delle ricompense nel caso di raggiungimento o non raggiungimento dell'obiettivo. Si è dimostrato

come inizializzare una Q(s,a) con valori tutti diversi da zero porti ad avere una maggiore velocità di convergenza della soluzione rispetto a Q(s,a) con tutti valori nulli.

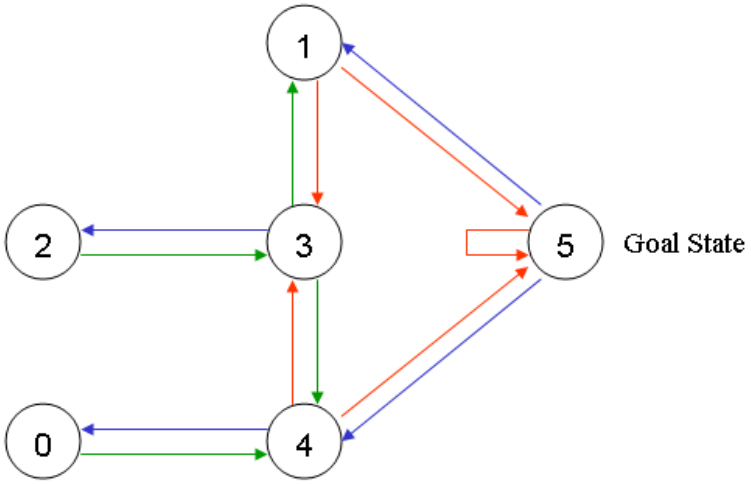
Esempio pratico: la ricerca di un percorso di evacuazione

L'esempio seguente si descrive la logica di funzionamento dell'algoritmo di Reinforcment Learning su un caso pratico. Viene illustrato passo dopo passo come un agente razionale, inizialmente "stupido" acquisisce ed utilizza la conoscenza acquisita con l'esperienza per individuare la migliore via di fuga da una abitazione.

Supponiamo di avere in seguente edificio costituito da 5 stanze collegate fra loro da porte come mostrato nella figura. Identifichiamo ogni stanza con un numero da 0 a 4. L'esterno dell'edificio può essere identificato come la stanza numero 5.

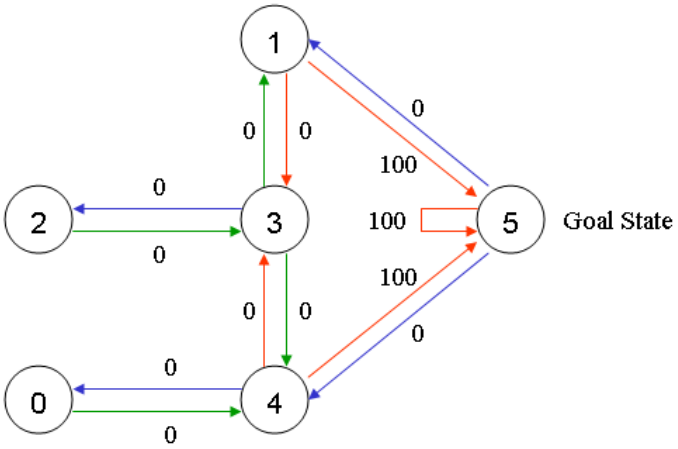


Possiamo rappresentare le stanze in un grafico, dove ogni stanza è rappresentata da un nodo e ogni porta di collegamento come un arco.



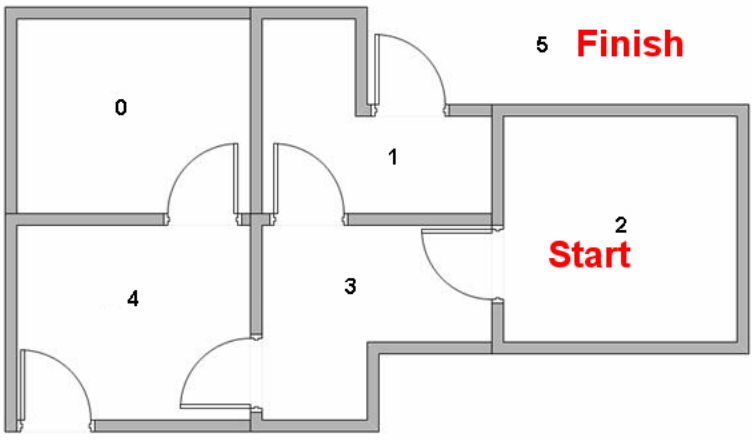
L'obiettivo dell'algoritmo è: posizionando l'agente in una qualsiasi delle stanze dell'abitazione, questo dovrà essere in grado di saper uscire dall'edificio. In altre parole, lo stato di uscita

dell'algoritmo o il goal è il raggiungimento della stanza o stato numero 5. Per impostare questo stato come goal, assoceremo un valore di ricompensa a ciascuna porta. Le porte che portano immediatamente all'obiettivo hanno una ricompensa istantanea di 100. Le altre porte, non direttamente collegate alla stanza obiettivo hanno una ricompensa zero. Poiché le porte sono bidirezionali (la porta 0 porta alla stanza 4 e 4 porta indietro a 0), a ogni stanza vengono assegnate due frecce. Ogni freccia contiene un valore di ricompensa, come mostrato di seguito:

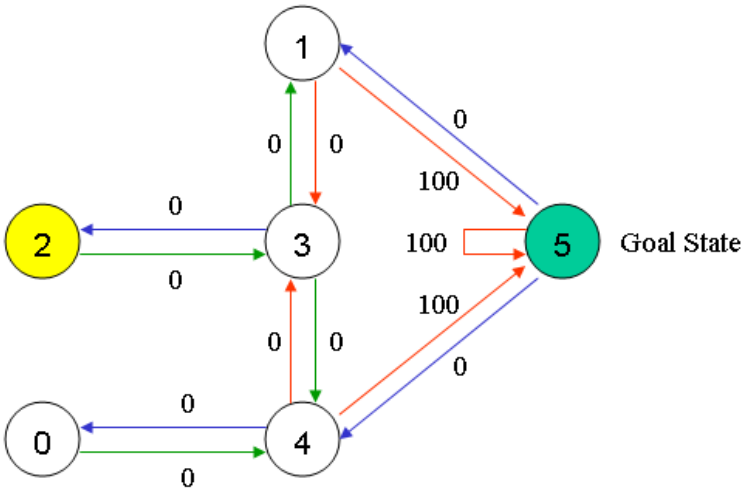


L'auto anello sullo stato 5 è una condizione che consente all'algoritmo di permanere in quello stato. In altri termini il nodo 5 viene chiamato "nodo assorbente ".

Consideriamo quindi il caso in cui l'agente "stupido" si trovi nella stanza 2 e voglia uscire:



Per fare ciò cercherà, inizialmente alla "cieca", di attraversare ogni porta che gli capita a tiro. Ma se ripercorrerà più volte tragitti diversi, si renderà conto ad un certo punto quale sia la strategia migliore per uscire da casa. Vediamo di seguito le varie possibilità:



Supponiamo che l'agente sia nello stato 2. Dallo stato 2, può passare allo stato 3 perché lo stato 2 è collegato a 3. Dallo stato 2, tuttavia, l'agente non può passare direttamente allo stato 1 perché non vi è alcuna porta di collegamento diretta 1 e 2. Dallo stato 3, può andare allo stato 1 o 4 o indietro a 2. Se l'agente si trova nello stato 4, le tre azioni possibili devono passare allo stato 0, 5 o 3. Se l'agente si trova nello stato 1, può passare allo stato 5 o 3. Dallo stato 0, può solo andare allo stato 4.

Possiamo inserire il diagramma di stato e i valori della ricompensa istantanea nella seguente tabella della ricompensa:

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

NOTA: In questo caso l'informazione della matrice R è una conoscenza dettagliata del sistema che noi trasferiamo alla macchina la sarà così capace di apprenderle ed inferire nuova conoscenza come le varie vie di fuga e la via di fuga ottimale partendo da ogni stanza. In molti casi reali queste informazioni sono frammentarie e parziali. Si conosce a volte soltanto il nodo obiettivo o in alcuni casi, come nei cicli di controllo di sistemi retro azionati, neanche quello. In ogni caso occorre quantificare, attraverso delle misure o delle stime, la bontà delle azioni che l'agente compie sul sistema.

L'agente razionale recepirà le informazioni contenute nella matrice R e li trasferirà all'interno della sua memoria interna attraverso una seconda matrice che chiameremo matrice "Q". Questa matrice rappresenterà il cervello del nostro agente, ovvero la memoria di ciò che l'agente ha imparato attraverso l'esperienza. Le righe della matrice Q rappresentano lo stato corrente dell'agente e le colonne rappresentano le possibili azioni che portano allo stato successivo (i collegamenti tra i nodi).

L'agente inizia senza sapere nulla sull'abitazione in cui viene collocato, in altre parole la matrice Q contiene inizialmente tutti valori nulli. In questo esempio assumiamo che il numero di stati sia noto, ovvero l'agente conosce di quante stanze è composta l'abitazione, ciò però non costituisce un vincolo, se non sapessimo quanti stati sono coinvolti, la matrice Q potrebbe iniziare con un solo elemento, dopo di chè ad ogni nuovo stato che si raggiunge si aggiunge semplicemente una riga alla matrice.

La regola di transizione dell'apprendimento Q è una formula molto semplice:

Q (stato, azione) = R (stato, azione) + Gamma * Max [Q (stato successivo, tutte le azioni)]

Rispetto alla regola più generale vista nel paragrafo precedente, in questo caso abbiamo posto per semplicità di presentazione, il coefficiente α pari ad 1. Assumendo alfa pari ad uno vuol dire che l'agente tiene in considerazione nell'aggiornamento di Q (ovvero della memoria) soltanto il valore atteso nella scelta della prossima azione, trascurando il valore precedentemente appreso. Diversamente, un valore di alfa pari a 0, annullerebbe questo secondo termine lasciando il valore di Q immutato.

Secondo questa formula, un valore assegnato a un elemento specifico della matrice Q, è uguale alla somma del valore corrispondente nella matrice R e al parametro di apprendimento Gamma, moltiplicato per il valore massimo di Q per tutte le possibili azioni nello stato successivo.

Il nostro agente virtuale imparerà attraverso l'esperienza le varie vie di fuga utilizzando un meccanismo di apprendimento che prende il nome di apprendimento non supervisionato. L'agente esplorerà da uno stato all'altro fino a raggiungere l'obiettivo. Chiameremo ogni esplorazione un episodio. Ogni episodio è costituito dall'agente che si sposta dallo stato iniziale allo stato obiettivo. Ogni volta che l'agente arriva allo stato obiettivo, il programma passa all'episodio successivo.

L'algoritmo può essere descritto dal seguente pseudocodice:

1. Set the gamma parameter, and environment rewards in matrix R.

2. Initialize matrix Q to zero.

3. For each episode:

Select a random initial state.

Do While the goal state hasn't been reached.

- Select one among all possible actions for the current state.
- Using this possible action, consider going to the next state.

- Get maximum Q value for this next state based on all possible actions.
- Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
- Set the next state as the current state.

End Do

End For

Possiamo affermare che ogni episodio equivale a una sessione di allenamento. In ogni sessione di allenamento, l'agente esplora l'ambiente (rappresentato dalla matrice R), riceve la ricompensa (se presente) fino a raggiungere lo stato obiettivo. Lo scopo della formazione è quello di migliorare la conoscenza del nostro agente (cervello), rappresentata dalla matrice Q. Più episodi di allenamento vengono effettuati e più la matrice Q risulta ottimizzata. Quando la matrice Q è stata migliorata, invece di esplorare e andare avanti e indietro nelle stesse stanze, l'agente troverà il percorso più veloce per raggiungere lo stato obiettivo.

Il parametro Gamma che viene inserito nell'algoritmo ha proprio questo obbiettivo. Se il valore di gamma è più vicino a zero, l'agente tenderà a considerare solo i premi immediati trascurando l'esperienza acquisita. Se Gamma è più vicino a uno, l'agente prenderà a considerazione le ricompense future con maggior peso, cioè è disposto a ritardare la ricompensa.

Conclusa la fase di apprendimento, la matrice Q può essere utilizzata per tracciare la sequenza di stati o azioni che dallo stato iniziale portano allo stato obiettivo.

L'algoritmo che trova le azioni con i più alti valori di ricompensa registrati nella matrice Q è il seguente:

1. Imposta lo stato corrente = stato iniziale.
2. Dallo stato corrente, trova l'azione con il valore Q più alto.
3. Imposta lo stato corrente = stato successivo.
4. Ripetere i passaggi 2 e 3 fino allo stato corrente = stato obiettivo.

L'algoritmo sopra riportato restituirà la sequenza di stati dallo stato iniziale allo stato obiettivo.

Per capire come funziona l'algoritmo Q-learning, analizzeremo alcuni episodi passo dopo passo.

Inizieremo impostando il valore del parametro di apprendimento Gamma = 0,8 e lo stato iniziale 1.

Inizializza la matrice Q come matrice zero:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Dalla seconda riga (stato 1) della matrice R ci sono due possibili azioni per lo stato corrente 1: vai allo stato 3 (r=0) o vai allo stato 5 (gli stati con -1 sono inammissibili). Selezionando casualmente di andare a 5 dove l'agente riceverà una ricompensa pari a 100.

$$R = \begin{array}{c} \\ \textbf{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \textbf{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Ora immaginiamo cosa succederebbe quando il nostro agente passa nello stato 5. Ha 3 azioni possibili: vai allo stato 1, 4 o 5. Pertanto la ricompensa che potrebbe aspettarsi scegliendo questo percorso sarà calcolata come:

Q (stato, azione) = R (stato, azione) + Gamma * Max [Q (stato successivo, tutte le azioni)]

Q (1, 5) = R (1, 5) + 0,8 * Max [Q (5, 1), Q (5, 4), Q (5, 5)] = 100 + 0,8 * 0 = 100

Poiché la matrice Q è ancora inizializzata su zero, Q (5, 1), Q (5, 4), Q (5, 5), sono tutti zero. Il risultato di questo calcolo per Q (1, 5) è 100 a causa della ricompensa istantanea di R (5, 1).

Lo stato successivo, 5, ora diventa lo stato corrente. Poiché 5 è lo stato obiettivo, abbiamo terminato un episodio. Il cervello del nostro agente ora contiene una matrice Q aggiornata come:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Per il prossimo episodio, iniziamo con uno stato iniziale scelto casualmente. Questa volta, abbiamo lo stato 3 come stato iniziale.

Guardando la quarta fila della matrice R, le possibili azioni sono 3: vai allo stato 1, 2 o 4. Selezionando casualmente di andare allo stato 1.

Ora immaginiamo cosa succederebbe quando il nostro agente passa nello stato 1. Guardando la seconda fila della matrice di ricompensa R, si hanno 2 azioni possibili: vai allo stato 3 o allo stato 5. Quindi, calcoliamo il valore Q:

Q (stato, azione) = R (stato, azione) + Gamma * Max [Q (stato successivo, tutte le azioni)]

Q (3, 1) = R (3, 1) + 0,8 * Max [Q (1, 2), Q (1, 5)] = 0 + 0,8 * Max (0, 100) = 80

Usiamo la matrice Q aggiornata dell'ultimo episodio. Q (1, 3) = 0 e Q (1, 5) = 100. Il risultato del calcolo è Q (3, 1) = 80 perché la ricompensa è zero. La matrice Q diventa:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Lo stato successivo, 1, ora diventa lo stato corrente. Ripetiamo il ciclo interno dell'algoritmo di apprendimento Q perché lo stato 1 che non è lo stato obiettivo.

Quindi, iniziando il nuovo loop con lo stato corrente 1, ci sono due possibili azioni: andare allo stato 3 o andare allo stato 5. Per sorteggio fortunato, la nostra azione selezionata è 5.

Ora, immaginando che siamo nello stato 5, ci sono tre possibili azioni: vai allo stato 1, 4 o 5. Calcoliamo il valore Q usando il valore massimo di queste possibili azioni.

Q (stato, azione) = R (stato, azione) + Gamma * Max [Q (stato successivo, tutte le azioni)]

Q (1, 5) = R (1, 5) + 0,8 * Max [Q (5, 1), Q (5, 4), Q (5, 5)] = 100 + 0,8 * 0 = 100

Le voci aggiornate della matrice Q, Q (5, 1), Q (5, 4), Q (5, 5) sono tutte zero. Il risultato di questo calcolo per Q (1, 5) è 100 a causa della ricompensa istantanea di R (5, 1). Questo risultato non cambia la matrice Q.

Poiché 5 è lo stato obiettivo, l'episodio termina con la seguente matrice aggiornata:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

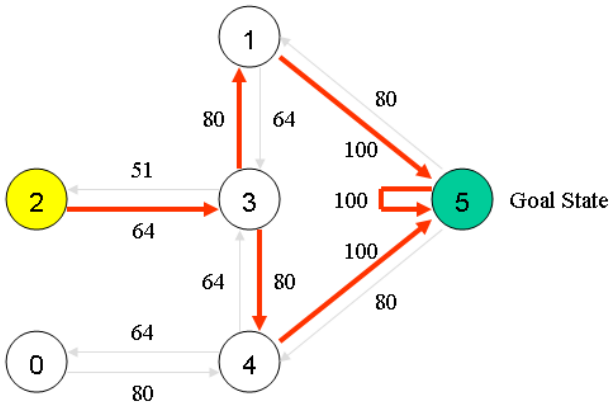
Se il nostro agente apprende di più attraverso ulteriori episodi, raggiungerà finalmente i valori di convergenza nella matrice Q come:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array}$$

Questa matrice Q, può quindi essere normalizzata (cioè; convertita in percentuale) dividendo tutte le voci diverse da zero per il numero più alto (500 in questo caso), avremo così una matrice di probabilità sulle transizioni:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

Quando la matrice Q si avvicina abbastanza a uno stato di convergenza, sappiamo che il nostro agente ha appreso i percorsi più ottimali per raggiungere lo stato obiettivo. Tracciare le migliori sequenze di stati che portano verso il goal adesso è semplice, basta seguire i collegamenti con i valori più alti in ogni stato.



Algoritmo SARSA

Una caratteristica fondamentale dell'algoritmo è l'utilizzo della funzione stato-azione Q(s,a) al posto della value function V(s). In particolare SARSA deve stimare Q π (s,a) per una determinata policy π e per ogni stato s e azione a . Questo avviene utilizzando la formula descritta in seguito:

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

Nell'algoritmo vengono considerate transizioni da una coppia stato-azione ad un'altra e l'aggiornamento avviene dopo ogni transizione indipendentemente se lo stato è terminale o no. Il nome SARSA deriva dal fatto che in realtà gli aggiornamenti si effettuano utilizzando una quintupla Q(s,a,r,s ′,a ′). Dove s e a rappresentano lo stato attuale e l'azione scelta nel passo corrente, invece s ′ e a ′ sono la nuova coppia stato-azione.
Il primo passo consiste nell'apprendere il valore di Q π per una determinata policy e successivamente, interagendo con l'ambiente, si può modificare e migliorare la policy.
I parametri γ e α rappresentano rispettivamente il discount factor e il learning rate. L'algoritmo SARSA converge con probabilità 1 ad una policy ottima e quindi ad un ottima funzione stato-azione Q(s,a) se le coppie stato-azione vengono visitate un numero infinito di volte.

Requisiti di progetto

Il progetto è stato svolto nell'ambito delle attività del progetto di ricerca PON ARS01_00836 denominato "COGITO – Sistema dinamico e cognitivo per consentire agli edifici di apprendere ed adattarsi".

La realizzazione del progetto consiste nel conseguimento dei seguenti macro obbiettivi:

1. formulazione del problema di schedulazione dei carichi domestici come problema di Reinforcement Learning;
2. sviluppo e test in ambiente simulato;
3. realizzazione di un prototipo con dispositivi IoT[8] ;
4. elaborato tecnico.

Relativamente al primo punto, viene richiesta la realizzazione di un modello[9] del problema utilizzando l'approccio o il paradigma del Reinforcement Learning[10]. Più in particolare si richiede di riformulare il noto problema di schedulazione[11] come un problema di RL attraverso l'uso di algoritmi afferenti alla classe denominata Temporal Difference (TD), ovvero algoritmi che basano l'apprendimento mediante differenza temporale e che adoperano la tecnica denominata bootstrap[12] per la stima corrente della funzione obiettivo[13]. Tra questi, si richiede di adoperare la versione base degli algoritmi Q-Learning e SARSA in quanto l'uso degli stessi consentirà la verifica del processo decisionale e dei risultati raggiunti.

Parametri

Per definire completamente un problema decisionale di scheduling è necessario definire e valorizzare alcuni parametri o vincoli del problema. Tali parametri potranno essere ritenuti fissi e predeterminati nella fase di realizzazione del modello in ambiente simulato, successivamente dovranno essere misurati ed elaborati dalla strumentazione richiesta nel progetto.
Di seguito una breve descrizione dei vincoli/parametri del problema ed i requisiti richiesti da progetto:

a) i carichi possono essere interattivi: direttamente controllati dall'utente, oppure differibili: gestiti da una unità di controllo. Il numero massimo di carichi differibili è pari a 6;
b) per ogni dispositivo è noto il parametro beta. Beta è un parametro scelto dall'utente ed indica l'intervallo temporale entro cui il carico deve essere attivato.
c) per ogni dispositivo è noto il parametro alfa. Alfa è un parametro scelto dall'utente ed indica il tempo di esecuzione di ogni carico. Tempo in cui l'elettrodomestico resta in funzione senza subire interruzioni;
d) la schedulazione dei carichi avviene in una finestra temporale di 24 ore suddivise in step temporali o decisionali di 15 minuti, ovvero 96 time stamp;

[8] Internet of Things (IoT) fa riferimento all'estensione alle cose dei benefici dell'uso di Internet finora limitati alle persone, permettendo agli oggetti di interagire con altri oggetti e quindi con le persone in modo sempre più digitale.

[9] rappresentazione formale e semplificata di un "pezzo" di realtà .

[10] è uno dei tre paradigmi principali dell'apprendimento automatico si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro. Questo tipo di apprendimento è solitamente modellizzato tramite i processi decisionali di Markov e può essere effettuato con diverse tipologie di algoritmi.

[11] problema decisionale che consiste nell'allocare risorse finite in modo tale che un dato obiettivo venga ottimizzato. Ambito della ricerca operativa che permette di risolvere problemi complessi e prendere decisioni tramite un modello matematico.

[12] è una tecnica statistica che permette di calcolare in modo approssimativo media e la varianza di uno stimatore senza conosce la distribuzione della statistica di interesse.

[13] la funzione obiettivo in un problema di scelta è una funzione di una o più variabili che esprime il fine in base al quale si intende effettuare la scelta.

e) è noto il profilo di consumo[14] dei dispositivi. Si possono utilizzare le informazioni contenute nei dati di targa[15] oppure da una stima ottenuta dagli appositi meter IoT[16];
f) la tariffa oraria dei prezzi dell'energia elettrica è nota sia nel caso di prelievo di energia dalla rete, sia nel caso di utilizzo di energia auto-prodotta da impianti a fonte rinnovabile[17];
g) la logica di attuazione che è la funzione obiettivo del problema decisionale, deve garantire un piano di schedulazione dei carichi tale da minimizzare la spesa energetica e ridurre i picchi di carico contenendoli sotto una soglia prefissata. Se la funzione obiettivo cercasse di minimizzare soltanto i costi, l'algoritmo cercherebbe di attivare tutti i carichi nello stesso istante di tempo. Ovvero nell'istante di minimo della curva dei costi di energia;

Per la realizzazione del prototipo è stato richiesto l'allestimento di un pannello dimostratore con le seguenti funzionalità:

- dispone di sei carichi controllabili[18] rappresentati da sei lampade con potenza differente;
- dispone di una presa di corrente per l'alimentazione dei carichi interattivi;
- consente l'attuazione dei singoli carichi attraverso tecnologia Wi-Fi;
- implementa una sottorete WLAN dedicata per gestire il traffico dati e garantire la sicurezza da attacchi informatici;
- dispone di un meter per ogni carico ed uno per l'intero sistema. I primi servono a caratterizzare il profilo energetico dei carichi controllati ed il secondo a misurare la potenza consumata dell'intero impianto. La finalità del secondo meter è quella di ricavare il prelievo energetico dei carichi interattivi e la dispersione dell'impianto;
- ingloba un server web per il controllo dei singoli dispositivi;
- ingloba un server web dedicato che espone i servizi di schedulazione verso le unità di controllo periferiche presenti nella rete (es. nel caso di utilizzo del sistema in un complesso abitativo condominiale dove sono presenti una o più unità di controllo per abitazione);
- integra un nodo per la misura e l'acquisizione dell'energia fotovoltaica incidente sul fabbricato.

Macro-obiettivo 1: Formulazione di un problema di schedulazione come problema di RL

Sintetizzando quando precedentemente esposto, l'obiettivo che ci prefiggiamo di raggiungere attraverso il metodo risolutivo adoperato è la generazione di una sequenza di attivazione di carichi

[14] Identifica l'andamento di potenza assorbita da un apparecchio durante il suo utilizzo.

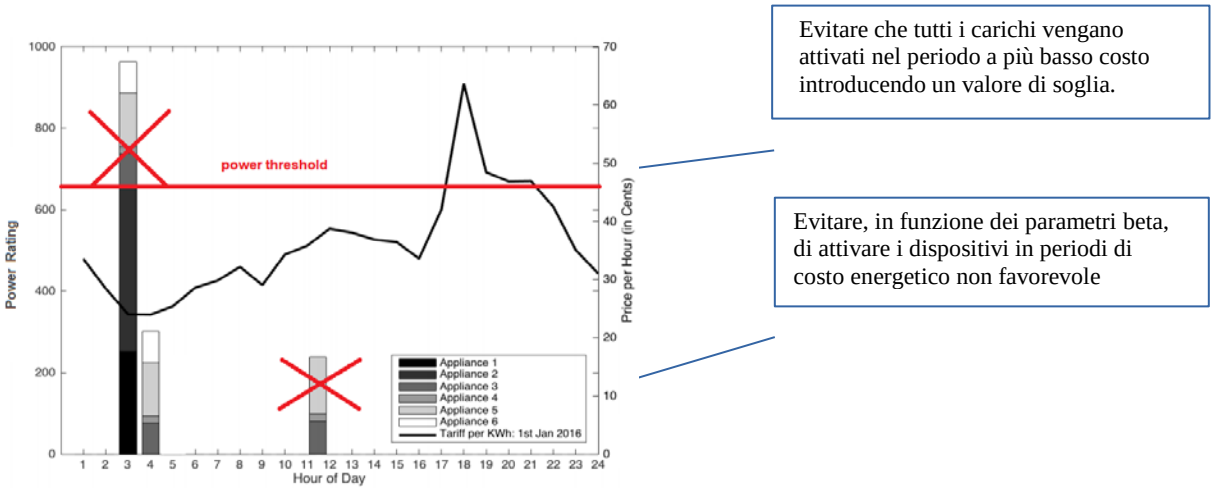
[15] Generalmente vengono riportati il consumo annuo normalizzato o la potenza nominale, che occorre convertire in potenza assorbita.

[16] In commercio sono sempre più diffusi attuatori IoT che eseguono anche un monitoraggio della potenza assorbita.

[17] Per gli impianti misti, composti da fonti rinnovabili e prelievo dalla rete elettrica, viene ricava una stima dell'andamento dei costi giornalieri, sottraendo al costo imposto dalla rete "il risparmio" ottenuto in previsione dell'energia prodotta dall'impianto fotovoltaico.

[18] Ovvero intermediati da un interruttore controllabile a distanza dal software di controllo.

domestici che vada a minimizzare i costi di consumo energetico e nel contempo vada ad evitare picchi di carico per l'attivazione simultanea di più dispositivi.



Per poter applicare gli algoritmi di RL al problema, innanzitutto occorre definire e strutturare il problema come un problema di MDP (Markov Decision Process), quindi occorrerà:

1) definire un insieme finito di stati;

2) definire un insieme finito di azioni di transizione tra gli stati;

3) definire una funzione di ricompensa che premi l'agente ad intraprende delle azioni;

Il problema così formulato deve godere della proprietà di Marckov, ovvero per ogni istante di tempo t T il sistema si troverà in un determinato stato appartenente ad un insieme numerabile di stati ∈ esaustivi e mutuamente esclusivi in cui l'agente potrà prendere la sua decisione guardando soltanto lo stato attuale e non lo stato precedente e/o l'evoluzione precedente effettuate.

Dalla valutazione del numero e della natura delle variabili che concorrono alla definizione della dinamica del sistema : stato dei dispositivi, tempo massimo di attivazione, tempo di esecuzione, potenza media dei dispositivi, potenza media di soglia, costo dell'energia, potenza prodotta dal fotovoltaico, numero di dispositivi da attivare e tempo di attivazione, occorre valutare accuratamente quali di queste concorrono alla definizione dello stato e quali concorrono alla definizione della funzione di reward per definire in tutte le sue parti il modello del sistema. Per superare questo scoglio, quello che generalmente si fa è considerare un sottoinsieme minimo delle variabili del sistema che concorrono alla definizione dello stato e successivamente inserire le variabili che vengono escluse all'interno della definizione della funzione di reward. Ricordiamo che quando si fa riferimento al modello del sistema, facciamo riferimento a tutte e tre gli elementi definiti poc'anzi: stato, azioni, reward. Concetto ben diverso invece è la definizione del problema di RL nella sua interezza. Quest'ultimo di fatti oltre alla definizione di un modello richiede la definizione di una policy. Tipicamente per la funzione di reward si trova una rappresentazione matematica che lega l'insieme delle variabili del problema con un numero reale. In questo lavoro viene data una definizione attraverso l'uso di pseudo-codice.

Definizione dello stato

Nel modello agente-ambiente, l'agente rappresentante "la parte decisionale" del sistema, deve essere in grado di immagazzinare un numero sufficiente di informazioni sul sistema tale da consentirgli di prendere decisioni in modo indipendente. Una decisione non è altro che l'azione migliore da scegliere tra quelle disponibili, tale da consentirgli il raggiungimento dell'obbiettivo prefissato.

L'azione è funzionale al raggiungimento di un nuovo stato del sistema. La scelta di inserire una variabile del sistema all'interno della rappresentazione dello stato è una scelta progettuale che ha forti implicazioni sul modello risultante. Il modello potrebbe non soddisfare a pieno la proprietà di Marckov oppure la cardinalità risultante degli stati potrebbe essere tale da rendere l'esecuzione degli algoritmi impraticabile. L'approccio quindi è aggiungere progressivamente un parametro del problema all'interno di una variabile di stato e capire se dal numero di informazioni inserite è possibile determinare l'intera evoluzione del sistema, mantenendo la proprietà di Marckov.

Nel nostro caso la scelta è stata di inglobare tre tipi di informazione nello stato. Queste rappresentano un sottoinsieme minimo di parametri del sistema in base al quale un'agente decisionale può intraprende delle scelte. Per poter effettuare una schedulazione temporale le informazioni minime da passare al decisore per noi sono le seguenti:

a) Il tempo di schedulazione. L'inserimento di questo parametro nello stato in realtà non è scontato. L'inserimento o l'esclusione determina il modus operandi dell'algoritmo. Se lo escludiamo, possiamo ipotizzare di effettuare delle schedulazioni "giornaliere" in cui si fissa un'ora di partenza ed una finestra temporale di esecuzione (ad esempio dalle 0:00 alle 24:00). Facendo questo presupposto ogni schedulazione inizierà allo step 0 (ore 0:00) e terminerà allo step 95 (ore 24:00). Quindi per collocare temporalmente un'attività all'interno della giornata sarà sufficiente moltiplicare la posizione dell'attività nella catena decisionale, per 15min. Il risultato dell'algoritmo sarà "il programma di schedulazione giornaliera". Tuttavia le sole informazioni di alfa e beta non sarebbero sufficienti per collocare i carichi sull'asse temporale. In questo caso andrà indicato una finestra di preferenza per l'attivazione, altrimenti l'algoritmo va a schedulare il carico in qualsiasi ora della giornata. I requisiti del progetto richiedono invece di realizzare un sistema real-time di supporto alle decisioni, pertanto è importante determinare sia l'ora di inserimento di una nuova schedulazione che l'ora di terminazione in quando queste due informazioni definiscono la finestra di attivazione voluta dall'utente. L'operatore potrà richiedere una nuova schedulazione in ogni momento della giornata ed il completamento della richiesta potrà esaurirsi nella stessa giornata o nel giorno successivo. Inoltre nella stessa giornata possono susseguirsi più programmi di schedulazione o possono addirittura sovrapporsi senza generare ambiguità o malfunzionamenti.

b) Stato di accensione dei dispositivi. Senza dubbio la variazione dello stato dei dispositivi è l'elemento minimo indispensabile che determina un passaggio di stato. Siamo in un ambiente dinamico, il valore impostato per un dispositivo può non essere quello effettivo. E' importante quindi rilevare lo stato di attivazione dei dispositivi prima di richiederne la schedulazione;

c) Stato desiderato o Richiesta utente. Senza questa informazione non saremmo in grado di definire lo stato obbiettivo o terminale dell'algoritmo;

Fatte queste scelte progettuali possiamo ora procedere ad analizzare alcuni aspetti salienti. La prima osservazione che si può fare è che non è possibile definire uno stato iniziale del modello da cui partire per addestrare l'algoritmo.

Se consideriamo come stato iniziale ad esempio (000,111,0) ed addestriamo un modello nel giorno x, lo stesso modello non sarà replicabile per il giorno y. Questo è dovuto al fatto che i valori di reward calcolati nel giorno x, sulla base dei valori di auto produzione, costo, consumo, etc. non sono identici e replicabili per il giorno y. In altri termini avendo variabilità temporale sui vincoli del sistema, ci ritroveremo la stessa variabilità sui reward e quindi sulla value function calcolata dall'algoritmo. Possiamo affermare che la matrice della conoscenza Q che definisce il modello del sistema è tempo variante. Questo ha le seguenti ripercussioni:

a) se le condizioni a contorno (variabili che concorrono alla definizione della funzione di reward) cambiano col tempo, allora sarà necessario replicare la fase di apprendimento per ogni richiesta. Questo è dovuto al fatto che gli algoritmi di RL acquisiscono conoscenza eseguendo più episodi di apprendimento partendo dallo stesso nodo iniziale, verso i nodi terminali considerando "immutate" le condizioni a contorno.

b) il percorso markoviano generato così come la matrice di conoscenza costruita per ricavarlo, non sono riutilizzabili per altre schedulazioni. La variabilità dei costi energetici, dei parametri di settaggio dei dispositivi, della potenza media assorbita, fanno sì che partendo da uno stesso nodo iniziale ma in giorni differenti, il percorso ottimo generato e la matrice di conoscenza Q cambiano.

c) non è possibile adoperare un'unica matrice di conoscenza Q per "memorizzare" più percorsi tra i nodi iniziali e quelli terminali. La matrice Q è utilizzata dagli algoritmi di RL sostanzialmente come un'area di memoria dove conservare i valori stimati della value function per la ricerca della politica ottimale (percorso ottimo). La stessa cella della matrice Q individuata da una coppia stato azione potrebbe essere interessata da più percorsi presenti nell'albero decisionale ed afferenti a nodi iniziali differenti. Questo porterebbe ad una sovrapposizione di valori.

Partendo da una prima rappresentazione dello stato definita come segue:

Possiamo determinare la quantità di memoria necessaria per il sistema e la complessità dell'algoritmo.

L'insieme degli stati avrà quindi una cardinalità pari ad: $S = \{96x2^{2n}\} = 393.216$

Mentre l'insieme delle azioni, definite come accensione o spegnimento di un dispositivo (0/1) avrà cardinalità pari ad: $A = \{2^n\} = 64$.

Pertanto la matrice Q avrà cardinalità pari ad: $S \times A = 25.165.824$

Anche se il numero degli stati attenuto non è un numero eccessivamente alto è possibile tuttavia effettuare altre semplificazioni.

Se consideriamo le ipotesi di progetto che se un carico viene attivato, questo rimarrà attivo ininterrottamente per un certo tempo prefissato allora lo “stato desiderato” e “stato corrente” possono essere ricondotti ad un’unica tupla che chiameremo “richiesta di attivazione”.

Il sistema dovrà quindi risolvere il problema di attivazione dei carichi senza preoccuparsi della disattivazione che avverrà in automatico (utilizzo di timer). Le azioni che consentiranno il passaggio da uno stato al successivo saranno quindi o istruzioni di attivazione o di differimento. All’interno dello stato a questo punto possiamo indicare con 0 il fatto che un dispositivo non è stato ancora attivato e con 1 dispositivo attivo o attivato.

Questa nuova rappresentazione dello stato non solo riduce la cardinalità degli stati da $96x2^{2n}$ a $96x2^{n}$, ma consente di scomporre il problema più generale di schedulazione di sei carichi, in sotto-problemi di minore complessità. Di fatti solo nel caso peggiore l’utente chiederà l'attivazione simultanea dei sei carichi, è molto più probabile che durante l’arco della giornata un utente chiederà l’attivazione dei carichi in tempi diversi ed al massimo di uno o due carichi per volta. Questa semplificazione del modello ci ha consentito di ricondurre il sistema, in un sistema real-time, poiché i tempi di risposta ora sono molto brevi così come l’ingombro di memoria richiesto per l'elaborazione.

Seguendo questa notazione, lo stato iniziale rappresentato dalla coppia (000,25) potrebbe identificare la richiesta di attivazione per il secondo, terzo e quarto dispositivo, oppure del primo, secondo e terzo dispositivo. È ovvio che nasce un problema di interpretazione.

Occorre quindi fare le seguenti ipotesi di lavoro:

a) il controllore che gestisce l’attuazione sui i dispositivi e riceve il comando dall’utente deve adottare una notazione posizionale per la gestione delle richieste verso lo scheduler. Nell’inviare la richiesta di schedulazione per lo stato iniziale (000,25) deve “ricordare” che il primo bit rappresenta il dispositivo 2 ed il terzo bit il dispositivo 4. Dopo di ché, ricevuta la sequenza di schedulazione, tramuta questa informazione in azioni di “attivazione” verso i relativi dispositivi, ovvero tramuta la sequenza di schedulazione in timer programmati di accensione e spegnimento per i singoli dispositivi;

b) per identificare uno stato terminale (nodo foglia), il cui raggiungimento da parte dell’algoritmo di RL determina la conclusione di un ciclo di apprendimento, non è sufficiente testare i bit presenti nello stato ma occorrerà testare anche i tempi di esecuzione di ogni attività. Informazione non contenuta nello stato. L’algoritmo quindi terminerà se tutte le richieste sono state attivate (bit ad 1 nello stato) ed inoltre tutti i parametri alfa sono a zero.

c) L’inserimento del tempo all’interno dello stato evita la creazione di auto-anelli. Il differimento di attivazione di un dispositivo non genera un auto-anello sullo stato ma bensì un passaggio tra lo stato (xxx,t) allo stato (xxx,t+1).

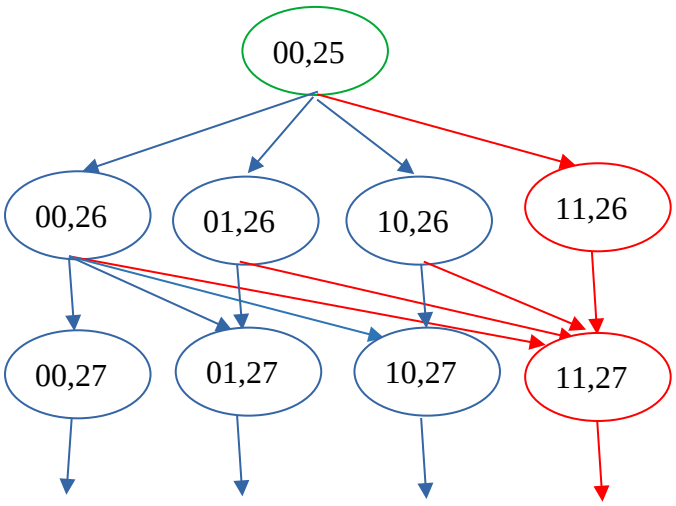
d) Lo spazio degli stati è un insieme ordinato secondo il parametro t (step). Pertanto la sequenza di schedulazione riporterà sempre valori di t crescente a partire dallo stato iniziale t0. In altri termini è possibile stabilire un ordinamento tra gli stati: (xxx,t+1) > (xxx,t).

e) Il tempo è ciclico, pertanto se deve essere processata una richiesta a partire dallo stato iniziale (000,94), raggiunto lo step temporale 95 l'elaborazione continuerà dallo stato con t =0. In altri termini è possibile stabilire questa proprietà: (xxx,0) > (xxx,95).

Albero decisionale – Marckov Decision Process Tree

Come anticipato nel paragrafo precedente, la matrice di conoscenza Q(s,a) che l'algoritmo di RL realizza per raggiungere una soluzione del problema assegnato, può essere rappresentata attraverso un albero decisionale. L'albero decisionale è una rappresentazione di un sistema costituito da uno stato iniziale, detto nodo di partenza, ed uno o più stati finali, detti nodo foglia. Le azioni o percorsi che collegano due stati del sistema sono detti archi. Il valore assegnato ad ogni cella della matrice Q(s,a), stimato dalla value function, rappresenta il peso di ogni arco. Trovare la politica ottima che risolve un problema di RL, equivale a trovare il percorso ottimo (percorso con peso massimo) tra il nodo iniziale ed il nodo finale, sull'albero decisionale. Gli algoritmi di TD-learning intervengono per "pesare" le scelte, ovvero gli archi del grafo, in funzione dei parametri che descrivono il sistema (Ci, Ki, αi e βi) al fine di determinare il percorso a guadagno massimo. Durante questa fase di ricerca si ipotizza che il sistema rimanga immutato, ovvero che le condizioni di partenza non vengono perturbate.

A questo punto è possibile rappresentare tutta la dinamica del sistema adoperando l'albero delle decisioni, chiamato **markov decision process (MDP)**.



Descrizione: l'MDP tree sopra rappresentato, descrive l'evoluzione di un sistema che opera una sequenza di attivazione di due dispositivi con l'obiettivo di minimizzare il costo e ridurre i picchi. Le azioni ammissibili per il nodo iniziale sono: attivare il primo dispositivo; attivare il secondo dispositivo; differire/attivare entrambi i dispositivi. L'algoritmo di RL assocerà un peso ad ognuno di questi archi (azioni) in funzione degli altri parametri del problema. Un nodo con valore (11,t) è terminale solo se tutti i parametri alfa (tempo di esecuzione attività) sono pari a zero. Raggiunto un nodo (11,t) l'algoritmo continuerà l'esecuzione passando dai nodi (11,t+1),(11,t+2),..fintanto che non verrà soddisfatta la condizione d'uscita. Il differimento delle attività non è infinito

(00,26),(00,27),(00,28),...etc., in quanto ad ogni step viene decrementato il parametro beta (tempo massimo di differimento) ed al superamento di tale vincolo interviene una penalità (reward negativo) che obbliga l'algoritmo a convergere. Il numero delle azioni ammissibili per ogni stato decresce in funzione dei dispositivi già attivati.

Alla luce delle considerazioni fatte sulla generazione del MDP tree, appare evidente che se costruiamo l'albero considerando soltanto le sole azioni ammissibili per ogni stato, allora possiamo constatare che più scendiamo in profondità verso i nodi foglia e più il numero di stati raggiungibili si dimezza. L'utilizzo di una funzione che restituisce l'insieme delle azioni ammissibili a partire da un dato stato ci consente di ridurre i tempi di convergenza dell'algoritmo e migliorare le risorse di memoria. La strategia di ridurre il campo di ricerca fa parte dalle **policy** del nostro algoritmo.

Scegliendo questo approccio si è notato che soltanto una parte della matrice Q veniva popolata dall'algoritmo. Ciò ci ha indotto a considerare l'uso di un albero come struttura dati, anziché una matrice portando importanti benefici all'intero sistema.

Relativamente alla ricerca delle azioni ammissibili per ogni stato, occorre fare una precisazione: vengono considerate inammissibili solo le azioni che tentano di attivare un dispositivo già attivato nella schedulazione corrente o in una precedente schedulazione. Non vengono inserite in questo filtro le azioni che portano in uno stato che viola un parametro del sistema (alfa,beta o costo). Questo perché ci potrebbero essere dei casi in cui si considerano accettabili anche soluzioni sub ottime che violano alcuni vincoli del problema a favore di tempi di risposta più brevi.

Reward function

Il terzo elemento da definire per completare la definizione di un modello del sistema è appunto la definizione della funzione di reward.

In molti problemi semplici definire un reward positivo per i nodi terminali è sufficiente per far convergere l'algoritmo di RL agli obiettivi prefissati. Tuttavia può essere fatto qualcosa di più. Si può in aggiunta associare un reward negativo a nodi intermedi. In questo caso l'algoritmo per ogni step decisionale in più che eseguirà per raggiungere l'obiettivo perderà ricompensa, pertanto sarà indirizzato a trovare, non un percorso qualsiasi verso il nodo terminale, ma il "**percorso minimo**". Valorizzando i nodi intermedi o gli archi intermedi si riesce pertanto a ***perseguire un secondo obiettivo***, in questo caso il percorso più breve tra lo stato iniziale e lo stato finale.

La definizione di reward per i nodi intermedi descrive una metodologia valida per far convergere l'algoritmo verso un nodo terminale nel rispetto dei vari sub-obbittivi prefissati all'interno dello spazio degli stati.

Nel caso di **problemi con più sub-obbiettivi** abbiamo quindi la necessità di stabilire delle ricompense o penalità per i ***nodi intermedi*** in modo da correggere il percorso decisionale dell'algoritmo verso i nodi foglia durante l'esecuzione indirizzandolo a raggiungere degli stati per noi desiderati ed evitarne altri che invece porterebbero alla violazione di alcuni vincoli.

Vediamo pertanto quali sono i vincoli del nostro problema o i sub-obbiettivi:

a) accendere tutti i dispositivi voluti l'operatore (macro obiettivo);

b) evitare picchi di consumo;

c) minimizzare il consumo medio giornaliero;

d) rispettare le finestre temporali di utilizzo - preferenza utente

Per rispettare questi vincoli è indispensabile pertanto definire dei reward intermedi che indirizzino l'agente di controllo a scegliere un percorso che ne tenga conto.

Per la risoluzione del problema scegliamo quindi di procedere in maniera incrementale. Consideriamo un obiettivo per volta trascurando gli altri vincoli e scegliamo una funzione reward che soddisfi il singolo obbiettivo senza però penalizzare la ricerca degli altri. Questo ultimo problema può essere affrontato definendo un reward positivo per le scelte corrette ed un ***reward neutro o nullo per le scelte considerate errate***. Dobbiamo tenere in conto la probabilità che ci potrebbero essere dei casi in cui non tutti i sub-obbiettivi potranno essere rispettati simultaneamente, pertanto mettere dei reward negativi sulla base di una valutazione di un singolo sub_obbiettivo, porterebbe a delle ripercussioni anche sulla ricerca di una soluzione che soddisfi gli altri requisiti.

La dinamica di interazione tra agente e ambiente, come schematizzato in pag.4, prevede che ad ogni step decisionale l'agente scelga, seguendo una certa politica, un'azione tra quelle ammissibili per lo stato, e successivamente attuando quell'azione sull'ambiente, riceve un feedback negativo o positivo conseguente a quell'azione. All'interno dell'algoritmo che descrive o interfaccia l'ambiente dovremmo sostanzialmente definire la seguente funzione:

state2, reward, done, info = env.step(action1)

Le informazioni presenti all'interno del modello "ambiente" sono complete ed esaustive, viceversa nella parte del modello "agente" si hanno a disposizione soltanto le informazioni di feedback di ritorno dalla funzione di reward. Per il calcolo del reward quindi possiamo attingere alle informazioni contenute nello stato attuale ed in quello precedente più il valore di tutte le variabili che concorrono alla definizione del problema.

La funzione di reward può essere quindi realizzata attraverso una sequenza di check sullo stato. Di seguito viene mostrato lo pseudocodice adoperato:

- ricompensa al raggiungimento dello stato terminale:

```
if((stato[0][i]==1 and alfa[i]<=0)
        reward += 1
```

Ricordiamo che lo stato è composto da due elementi: S(disp,timestamp), dove disp è una tupla binaria che descrive lo stato di attivazione dei dispositivi. Ad esempio 100, individua lo stato in cui è stato attivato il primo dispositivo (sugli N presenti sul controllore). Il valore 1 identifica il fatto che è stata eseguita una richiesta di attivazione, ma non ci dà informazioni sullo stato effettivo. Verificando il valore di alfa (tempo di esecuzione)

ricaviamo il tempo trascorso e possiamo concludere se l'esecuzione del dispositivo è terminata oppure no.

- Ricompensa per l'attivazione di un dispositivo avvenuta in condizioni energetiche favorevoli:

```
if(statoprec[i]==0 and stato[i]==1):
    if(Cdisp[i,t] == min(Cdisp[i])):
        reward += 1
    else:
        reward -= 1
```

In questa funzione notiamo la ricerca su un vettore di costi: *min(Cdisp[i]).*

Questo vettore è ottenuto moltiplicando la potenza media stimata del dispositivo, il tempo di attività del dispositivo (alfa) ed il costo orario stimato di energia per quella fascia oraria.

Il costo orario stimato di energia è frutto anch'esso di una elaborazione ottenuta a partire dai dati raccolti dalla produzione fotovoltaica e dell'andamento dei costi giornalieri di energia fissati dall'operatore.

Il primo check effettuato nella funzione va a verificare se l'azione ha generato l'attivazione di un dispositivo. In caso affermativo si va a verificare se il costo generato dall'attivazione del dispositivo nell'intervallo "t - t+alfa" corrisponde al costo minimo previsto per le prossime 24 ore e per lo stesso intervallo di attivazione. Avendo presupposto il tempo ciclico, l'attivazione di un dispositivo può essere differita nel giorno successivo, pertanto la ricerca viene eseguita su tutto il vettore.

- Verifica del rispetto del parametro beta, tempo massimo di differimento:

```
if (stato[i] == 0 and beta[i] <=0):
    reward -= 1
```

Si effettua il check solo sul parametro beta e non su alfa in quanto quest'ultimo è gestito dal controllore attraverso l'uso di timer e quindi non è oggetto di decisione. Il controllo in questo caso consiste nel verificare se lo stato raggiunto dall'azione porta il dispositivo "i" in condizioni da non poter soddisfare il parametro beta.

- Verifica della soglia di potenza:

```
if(stato[i]==1 and alfa[i]>0):
```

```
            potenza += potmed[i]
        if(potenza >soglia[t]):
            reward -= 1
```

In questo caso occorre mantenere in memoria il valore della potenza impegnata dai dispositivi “accesi” e non “attivi” nella corrente schedulazione. La differenza è stata ampiamente discussa in precedenza.

Questo valore va confrontato col valore di soglia. La soglia in questo caso non è un valore numerico fissato dall’utente ma viene calcolato sottraendo alla potenza massima consentita dall’impianto (es. 3kw) , la potenza non gestita (carichi non differibili) con la potenza “impegnata” da altri dispositivi differibili schedulati precedentemente e non presenti nell’attuale schedulazione.

Macro-obiettivo 2: Sviluppo e test in ambiente simulato

Alla luce delle considerazioni fatte nei paragrafi precedenti risulta evidente che molte delle informazioni necessarie per calcolare la funzione di reward sono frutto una elaborazione di dati precedentemente raccolti nel sistema. I dati necessari per l’elaborazione sono:

- potenza media dei dispositivi controllati;
- potenza media di tutti i carichi interattivi;
- potenza immessa nel sistema dalla produzione fotovoltaica;
- costo dell’energia prelevata dalla rete;
- programmi di schedulazione in corso di esecuzione.

Questi dati, quindi devono essere raccolti, memorizzati ed elaborati per poter poi essere utilizzati dall’algoritmo. In questa fase diamo per assodato che questi siano disponibili e stazionari.

In questa parte di elaborato vengono date le nozioni di base di uno dei più utilizzati ambienti di sviluppo e di addestramento per gli algoritmi di RL, OpenAI Gym. Successivamente vengono presentati i primi risultati prodotti in ambiente simulato. A tal fine sono stati analizzati due algoritmi: SARSA e Q_Learnig sulla base di tre casi di studio, utilizzando dati assegnati staticamente.

Definizione del package gym_scheduler

Una delle sfide maggiori del Reinforcement Learning è disporre di ambienti di test funzionanti in cui poter addestrare l’agente. Ecco che entrano in gioco gli ambienti di simulazione (simulated environment) in cui poter allenare la nostra intelligenza artificiale.

Nel concreto la libreria Gym è una collezione di test problems, chiamati environemnts (ambienti) che possono essere impiegate come “palestre” di allenamento per la ricerca e lo sviluppo di nuovi algoritmi di reinforcement learning.
Gli environments condividono un’interfaccia comune, questo significa che possiamo scrivere algoritmi generali senza doverli adattare ai diversi ambienti.
La classe env è la classe principale dell’ambiente OpenAI. Incapsula un ambiente con le rispettive dinamiche. Un ambiente in gym può essere parzialmente o completamente osservabile.
I principali metodi API che gli utenti di questa classe devono conoscere sono:

- reset
- step
- render

La classe env inoltre possiede i seguenti attributi:

- action_space: insieme delle azioni possibili sull’ambiente;
- observation_space: insieme degli stati definiti dall’ambiente;

la funzione step accetta come parametro un action e restituisce 4 valori:

observation, reward, done, info = env.step(action)

- observation (oggetto): è lo stato restituito dall'ambiente a seguito di una specifica azione. Rappresenta l’osservazione dell'ambiente dopo aver eseguito un’azione. Ad esempio, i dati dei pixel di una telecamera, gli angoli e le velocità dei giunti di un robot o lo stato della tavola in un gioco da tavolo.
- reward (float): rappresenta la ricompensa raggiunta dall'azione precedente compiuta. Il suo valore varia a seconda dell’azione e l'obiettivo è sempre quello di raggiungere la ricompensa massima totale.
- done (booleano): tale valore serve ad indicare se l'episodio è terminato. Se il valore restituito è TRUE, allora si può ripristinare nuovamente l'ambiente per dare il via ad un nuovo episodio di apprendimento.
- Info (dict): rappresentano informazioni diagnostiche utili per il debug. Dict è un dizionario composto da coppie (chiave,valore). A volte può essere utile per l'apprendimento restituire altre informazioni otre a quelle viste ad esempio, potrebbe essere utile restituire anche le probabilità grezze date dall'ultimo cambiamento di stato.

La funzione reset: è un metodo che verrà chiamato per ripristinare periodicamente l'ambiente su uno stato iniziale;

La funzione render: metodo che può essere chiamato periodicamente per visualizzare o stampare una rappresentazione dell'ambiente. Questo metodo potrebbe essere definito attraverso una semplice istruzione print oppure come il rendering di un ambiente 3D usando openGL.

Per creare un nuovo ambiente da utilizzare per addestrare la nostra intelligenza artificiale, occorre creare un repository contenente un package installabile nel sistema.

L'istruzione per l'istallazione è la seguente:

```
cd  gym-scheduler

pip install -e
```

Dopo aver istallato il package sarà possibile utilizzarlo nell'ambiente gym attraverso l'istruzione:

```
import gym

import gym_scheduler

env = gym.make('SchedulerEnv-v0')
```

NOTA: il repository è definito dal nome gym-nomeambiente, ovvero dal termine gym seguito dal trattino mentre il package dal nome gym_nomeambiente, ovvero dal termine gym seguito da underscore.

Secondo la documentazione rilasciata da OpenAI gym, la struttura del repository deve rispettare la seguente struttura:

```
gym-scheduler/

README.md

  setup.py

  gym_scheduler/

    __init__.py

    envs/

      __init__.py

      scheduler_env.py
```

All'interno della struttura troviamo la classe scheduler_evn.py. Classe che contiene la dinamica dell'ambiente. Gli altri script che troviamo nella struttura sono metodi standard che consentono l'istallazione dell'ambiente in gym.

Per completezza descriviamo ogni singolo elemento del nostro package:

```
gym-scheduler/gym-setup.py

from setuptools import setup

setup(name='gym_scheduler',

      version='0.0.1',

      install_requires=['gym']  # And any other dependencies foo needs

)
```

```
gym-scheduler/gym-scheduler/__init__.py
from gym.envs.registration import register
register(
    id='scheduler-v0',
    entry_point='gym_scheduler.envs:SchedulerEnv',
)
gym-scheduler/gym-scheduler/envs/__init__.py
from gym_scheduler.envs.scheduler_env import SchedulerEnv
gym-scheduler/gym-scheduler/envs/scheduler_env.py
import gym
from gym import error, spaces, utils
from gym.utils import seeding
class SchedulerEnv(gym.Env):
  metadata = {'render.modes': ['human']}
  def __init__(self):
  def step(self, action):
  def reset(self):
  def render(self, mode='human'):
  def close(self):
```

Testing Google Colab

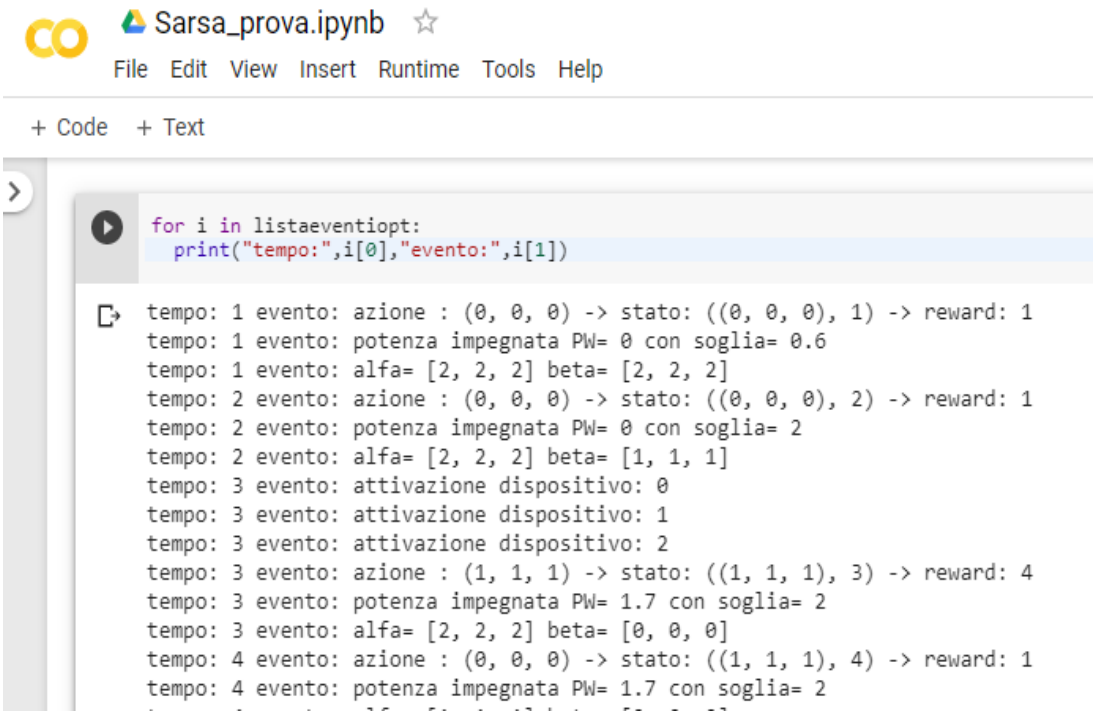
Realizzato il modello di ambiente ci occorre a questo punto definire ed installare un ambiente per poter eseguire le nostre simulazioni. A questo scopo si è deciso di utilizzare l'ambiente Google Colab, una piattaforma che ci permette di eseguire codice direttamente sul Cloud, sfruttando la potenza di calcolo fornita da Google. Per sfruttare le funzionalità di tale piattaforma, tutto ciò di cui abbiamo bisogno è un account Google, mediante il quale potremo effettuare il login ed avere accesso al servizio.

Per eseguire codice, Google Colab sfrutta i cosiddetti Jupyter Notebook. Questi non sono altro che documenti interattivi nei quali possiamo scrivere (e quindi eseguire) il nostro codice. Più precisamente, tali documenti permettono di suddividere il nostro codice in celle, ognuna delle quali può contenere anche del testo informativo.

L’uso di tali notebook è abbastanza comodo per chi si occupa di data science e machine learning. Tramite un unico documento, è infatti possibile sia eseguire tutti gli step di un processo di analisi o processing, sia descriverne il comportamento in linguaggio naturale.

La possibilità di dividere il codice in celle ha facilitato notevolmente la fase di sviluppo e testing. Separando in celle diverse le variabili del sistema, il codice che descrive l’ambiente ed i singoli algoritmi di RL o le singole funzioni che costituiscono il sistema è stato possibile verificarne il comportamento ed eliminare gli errori.

Un esempio dell’ambiente è mostrato di seguito:



Analisi dei risultati e casi di studio

Di seguito verranno presentai tre casi di studio reputati significativi al fine di testare l’algoritmo. Nell’ambiente di simulazione verranno istanziati due oggetti, uno relativo alla classe che definisce l’ambiente ed uno relativo alla classe che definisce l’algoritmo di controllo.

Per istanziare un oggetto della classe ambiente abbiamo bisogno di passare al costruttore una serie di parametri che andremo a descrivere di seguito:

- numerodispositivi – intero compreso nell'intervallo 1-6 che indica la quantità di dispositivi di cui si è richiesta l'attivazione;
- timestamp – intero compreso tra 0-11 oppure 0-95 che rappresenta l'intervallo temporale in cui arriva la richiesta di schedulazione ;
- potenzadispositivi – array di numeri reali con lunghezza compresa tra 1-6 rappresentante la potenza media dei dispositivi di cui si è richiesta l'attivazione;
- soglia – array di numeri reali di lunghezza pari 12 oppure 96 rappresentante il valore di soglia di potenza impostato nell'arco della giornata. Il valore di soglia non è fissato per l'intera giornata ma viene rielaborato sottraendo il valore di potenza dei dispositivi già attivati al momento della richiesta e la potenza consumata dai dispositivi interattivi;
- potenzaimpegnata – matrice data da [numerodispositivi,timestamps] che contiene nel dettaglio i valori di potenza sottratti da ogni dispositivo di cui si è richiesta l'attivazione. Diversamente dalla soglia in questa vengono forniti i valori di potenza dei soli dispositivi da attivare che potrebbero essere già accesi al momento della richiesta;
- alfa – array di interi compresi tra 1-11 o 1-95 che descrive il tempo di utilizzo di un dispositivo;
- beta – array di interi compresi tra 1-11 o 1-95 che descrive il tempo di differimento di accensione di ogni dispositivo;
- costo_orario – array riportante i valori del costo dell'energia durante l'arco della giornata;

Allo stesso modo al costruttore della classe di controllo andremo a passare i seguenti parametri che definiscono la politica di apprendimento:

- epsilon – numero reale compreso tra 0 ed 1. Un valore di epsilon=0.8 significa che inizialmente la politica di esplorazione verrà scelta con percentuale del 80% rispetto alla politica di learning.
- total_episodes – Intero che indica il numero di episodi su cui deve ciclare l'algoritmo per concludere la fase di addestramento;
- max_steps – Intero che limita il numero di iterazioni che esegue l'algoritmo nella ricerca di un nodo terminale;
- alpha - tasso di apprendimento. Numero reale compreso tra 0 (algoritmo non apprende nulla) ed 1 (l'algoritmo considera solo l'osservazione più recente)
- gamma – numero reale compreso tra 0 ed 1. Indica il tasso di ricompensa allo step successivo. Il valore 0 considera solo la ricompensa attuale, mentre il valore 1 permette di cercare ricompense a lungo termine.

Primo caso di studio

Sottoponiamo l'algoritmo ad una richiesta di attivazione per tre dispositivi che risultino già accesi al momento della richiesta e che le condizioni di costo energetico siano favorevoli. Ci aspettiamo che l'algoritmo scheduli l'attivazione dei tre carichi a conclusione della schedulazione precedente. Al fine di verificare tutte le scelte decisionali consideriamo una suddivisione del tempo in intervalli di due ore e quindi con una suddivisione della durata della giornata su 12 timestamp. Supponiamo che la richiesta arrivi al timestamp T=0 e si adotti una fascia di costo bioraria (secondo i valori reali di costo imposti da Enel distribuzione al momento del test). Impostando i seguenti parametri di partenza:

```
potenzadispositivi = [0.5,0.9,0.5]
timestamp = 0
numerodispositivi = 3
soglia=[0.1,0.6,2,2,2,2,2,2,2,2,2,2]
potenzaimpegnata=np.array([[0.5,0.5,0,0,0,0,0,0,0,0,0,0],
                           [0.9,0.9,0,0,0,0,0,0,0,0,0,0],
                           [0.5,0,0,0,0,0,0,0,0,0,0,0]])

alfa = [2,2,2]
beta = [3,3,3]
costo_orario [0.0566, 0.0566, 0.0566, 0.0566, 0.0630, 0.0630, 0.0566, 0.0566, 0.0566, 0.0566,
0.0566, 0.0566]
```

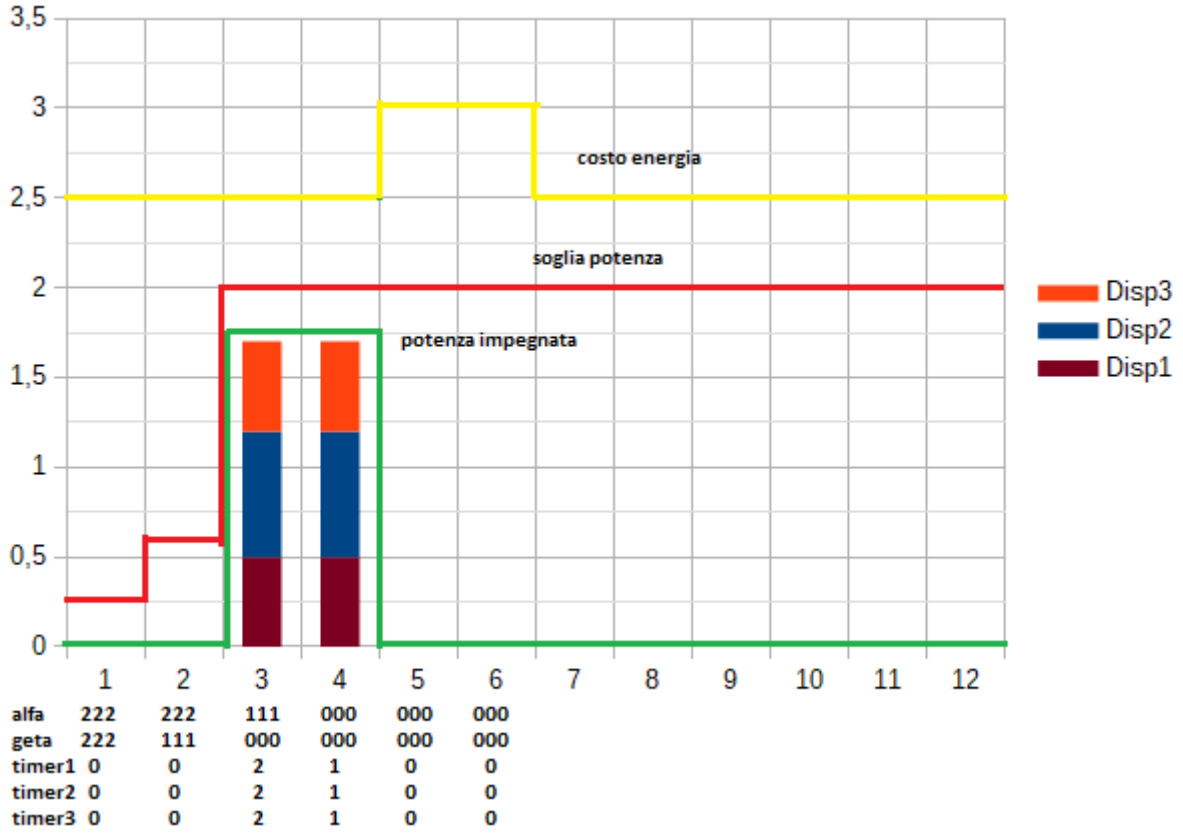
Il risultato ottenuto dall'algoritmo SARSA dopo 10 iterazioni di addestramento è stato:

path: [((0, 0, 0), 0), ((0, 0, 0), 1), ((0, 0, 0), 2), ((1, 1, 1), 3), ((1, 1, 1), 4), ((1, 1, 1), 5)] con reward: 9 valuefunction 78.88

```
((0, 0, 0), 0), ((0, 0, 0), 1)
tempo: 1 evento: azione : (0, 0, 0) -> stato: ((0, 0, 0), 1) -> reward: 1
tempo: 1 evento: potenza impegnata PW= 0 con soglia= 0.6
tempo: 1 evento: alfa= [2, 2, 2] beta= [2, 2, 2]
((0, 0, 0), 1), ((0, 0, 0), 2)
tempo: 2 evento: azione : (0, 0, 0) -> stato: ((0, 0, 0), 2) -> reward: 1
tempo: 2 evento: potenza impegnata PW= 0 con soglia= 2
tempo: 2 evento: alfa= [2, 2, 2] beta= [1, 1, 1]
((0, 0, 0), 2), ((1, 1, 1), 3)
tempo: 3 evento: attivazione dispositivo: 0
tempo: 3 evento: attivazione dispositivo: 1
tempo: 3 evento: attivazione dispositivo: 2
tempo: 3 evento: azione : (1, 1, 1) -> stato: ((1, 1, 1), 3) -> reward: 4
tempo: 3 evento: potenza impegnata PW= 1.7 con soglia= 2
tempo: 3 evento: alfa= [2, 2, 2] beta= [0, 0, 0]
((1, 1, 1), 3), ((1, 1, 1), 4)
tempo: 4 evento: azione : (0, 0, 0) -> stato: ((1, 1, 1), 4) -> reward: 1
tempo: 4 evento: potenza impegnata PW= 1.7 con soglia= 2
tempo: 4 evento: alfa= [1, 1, 1] beta= [0, 0, 0]
((1, 1, 1), 4), ((1, 1, 1), 5)
tempo: 5 evento: timer scaduto per: 0
tempo: 5 evento: timer scaduto per: 1
tempo: 5 evento: timer scaduto per: 2
tempo: 5 evento :----schedulazione terminata allo step: 5----------
tempo: 5 evento: azione : (0, 0, 0) -> stato: ((1, 1, 1), 5) -> reward: 2
tempo: 5 evento: potenza impegnata PW= 0.0 con soglia= 2
```

tempo: 5 evento: alfa= [0, 0, 0] beta= [3, 3, 3]

Attraverso la seguente rappresentazione grafica si può verificare come l'algoritmo individua correttamente lo slot temporale di collocazione dei tre carichi nel rispetto dei parametri inseriti.



Contestualmente al calcolo del percorso ottimo ricavato a partire dai valori stimati nella matrice Q, abbiamo aggiunto il calcolo del cammino con massimo reward cumulativo. Questo ci ha consentito di effettuare i dovuti confronti e settaggi dei valori di settaggio dell'algoritmo. In effetti per questo caso di studio sono emersi due cammini con reward pari a 9. Il secondo cammino con reward cumulativo massimo che potrebbe ricavare l'algoritmo è il seguente:

path: [((0, 0, 0), 0), ((0, 0, 0), 1), ((0, 0, 0), 2), ((1, 1, 0), 3), ((1, 1, 0), 4), ((1, 1, 0), 5), ((1, 1, 0), 6), ((1, 1, 1), 7), ((1, 1, 1), 8), ((1, 1, 1), 9)] con reward: 9 **valuefunction 57.96**

Questo secondo percorso ottenuto pospone l'accensione del terzo dispositivo al timestamp 7, violando il vincolo beta (fissato a 3) mantenendo comunque i requisiti di minimo costo.

Lasciando il numero di iterazioni di apprendimento a 10, una volta su tre verrà prodotto il cammino errato. Aumentando il numero di iterazioni a 100, la probabilità di avere il cammino scorretto cala notevolmente.

Una analisi più dettagliata degli step decisionali è mostrato di seguito:

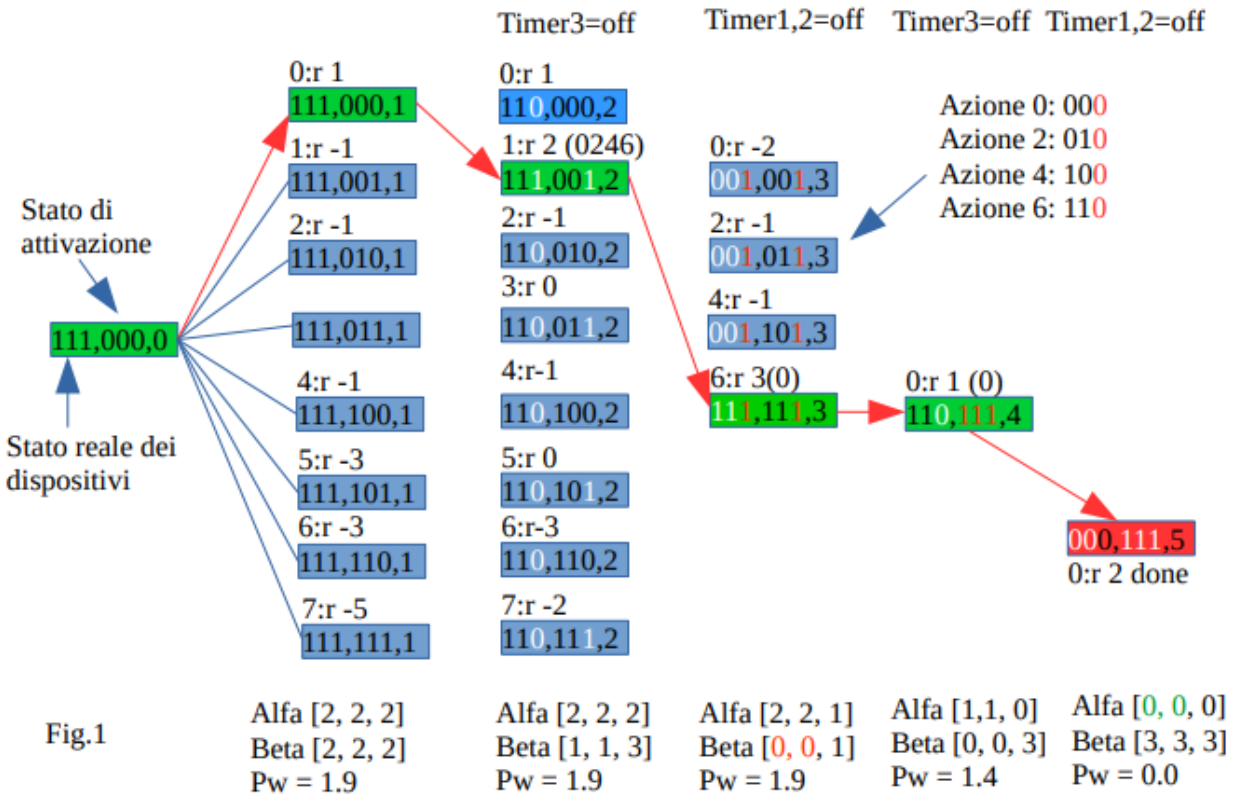


Fig.1

Al primo step l'unica azione a reward positivo è lo "sleep" in quanto le altre tentando di accendere i dispositivi già attivati nella schedulazione precedente ottengono reward negativo. Tutte le combinazioni soddisfano la condizione di costo e di soglia di potenza in quando siamo in zona a basso costo e la potenza impegnata è 1.9. Al secondo step è "previsto" lo spegnimento del terzo elemento per cui l'unica azione premiata e quella che lo riaccende. Al terzo step, escludendo il terzo elemento già attivato premiano l'azione 6 perché è l'unica che soddisfa i parametri beta. Al quinto step avremo tutti gli alfa a zero che identifica l'azione d' uscita.

Secondo caso di studio

In questo secondo test sottoponiamo l'algoritmo alla gestione di una richiesta di attivazione di tre dispositivi in condizioni di costo energetico favorevole. Ci aspettiamo che l'algoritmo scheduli l'accensione dei tre carichi differendone l'attivazione solo in funzione del vincolo della soglia di potenza. Impostiamo gli stessi parametri del primo caso di studio:

potenzadispositivi = [0.5,0.7,1.8]

timestamp = 0

numerodispositivi = 3

soglia=[2,2,2,2,2,2,2,2,2,2,2,2]

potenzaimpegnata=np.array([[0,0,0,0,0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0,0,0,0,0],

[0,0,0,0,0,0,0,0,0,0,0,0]])

alfa = [3,2,2]

beta = [1,3,3]

costo_orario [0.0566, 0.0566, 0.0566, 0.0566, 0.0630, 0.0630, 0.0566, 0.0566, 0.0566, 0.0566, 0.0566, 0.0566]

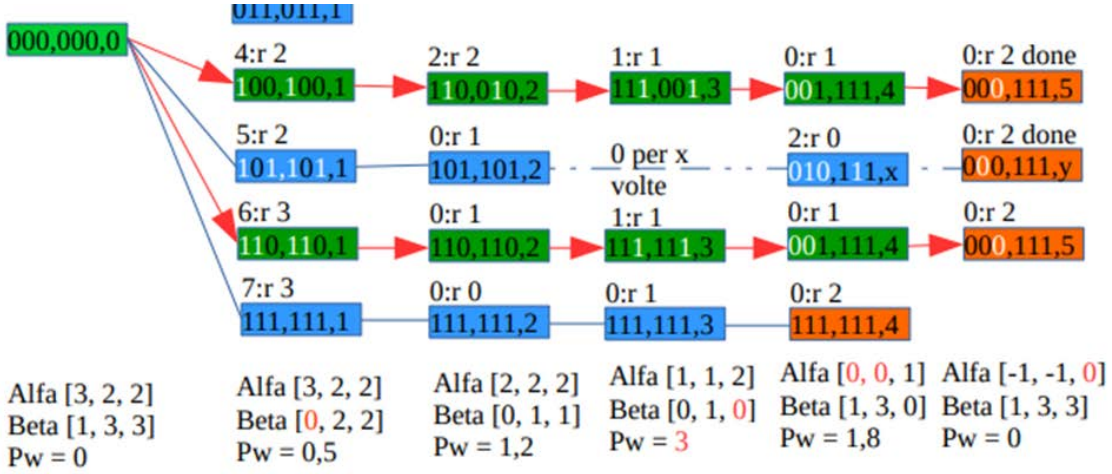
Diversamente al primo caso di studio dove con sole 10 iterazioni sia l'algoritmo SARSA che Q_Learning riusciva a trovare il percorso ottimo, in questo caso dobbiamo raggiungere 1000 iterazioni affinché l'algoritmo Q_Learning dia il percorso ottimo.

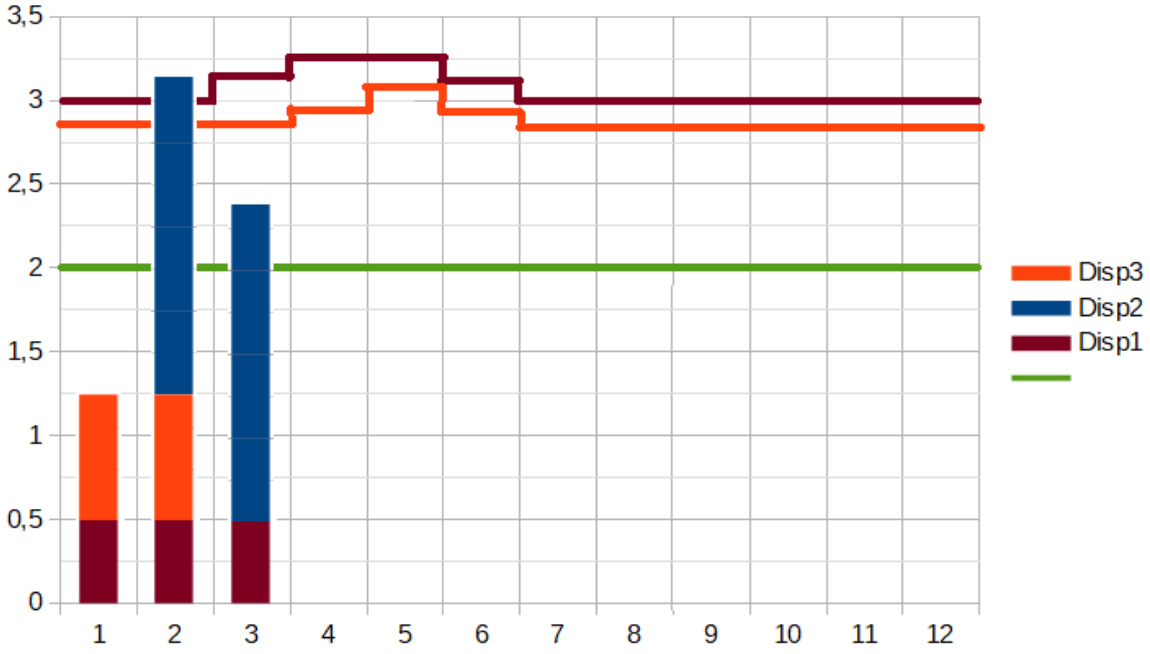
path: [((0, 0, 0), 0), ((1, 0, 0), 1), ((1, 1, 1), 2), ((1, 1, 1), 3), ((1, 1, 1), 4)] con reward: 6 value function 47.25 con 10 iterazioni

path: [((0, 0, 0), 0), ((1, 1, 0), 1), ((1, 1, 1), 2), ((1, 1, 1), 3), ((1, 1, 1), 4)] con reward: 6 valu efunction 70.39 con 1000 iterazioni

tempo: 1 evento: attivazione dispositivo: 0
tempo: 1 evento: attivazione dispositivo: 1
tempo: 1 evento: azione : (1, 1, 0) -> stato: ((1, 1, 0), 1) -> reward: 3
tempo: 1 evento: potenza impegnata PW= 1.2 con soglia= 2
tempo: 1 evento: alfa= [3, 2, 2] beta= [0, 2, 2]
tempo: 2 evento: attivazione dispositivo: 2
tempo: 2 evento: azione : (0, 0, 1) -> stato: ((1, 1, 1), 2) -> reward: 2
tempo: 2 evento: potenza impegnata PW= 3.0 con soglia= 2
tempo: 2 evento: alfa= [2, 1, 2] beta= [0, 2, 1]
tempo: 2 evento: superata la soglia di potenza PW= 3.0
tempo: 3 evento: timer scaduto per: 1
tempo: 3 evento: azione : (0, 0, 0) -> stato: ((1, 1, 1), 3) -> reward: 1
tempo: 3 evento: potenza impegnata PW= 2.3 con soglia= 2
tempo: 3 evento: alfa= [1, 0, 1] beta= [0, 3, 1]
tempo: 3 evento: superata la soglia di potenza PW= 2.3
tempo: 4 evento: timer scaduto per: 0
tempo: 4 evento: timer scaduto per: 2
tempo: 4 evento: azione : (0, 0, 0) -> stato: ((1, 1, 1), 4) -> reward: 2
tempo: 4 evento: potenza impegnata PW= 0.0 con soglia= 2
tempo: 4 evento: alfa= [0, -1, 0] beta= [1, 3, 3]
tempo: 4 evento: ----------schedulazione terminata allo step: 4----------

Analizzando l'albero di decisioni vediamo che sono presenti due percorsi a pari valore di reward cumulativo massimo. La scelta che prevarica in entrambi i casi è quella di accendere nel rispetto di beta e trascurando il superamento della soglia di potenza.





Terzo caso di studio

In questo terzo test sottoponiamo l'algoritmo alla gestione di una richiesta di attivazione di tre dispositivi in condizioni di costo energetico sfavorevole. Ci aspettiamo che l'algoritmo scheduli l'accensione dei tre carichi differendone l'attivazione in un periodo della giornata con costi minimi e nel rispetto del vincolo della soglia di potenza. I parametri impostati in questo caso saranno:

potenzadispositivi = [0.5,0.7,1.8]

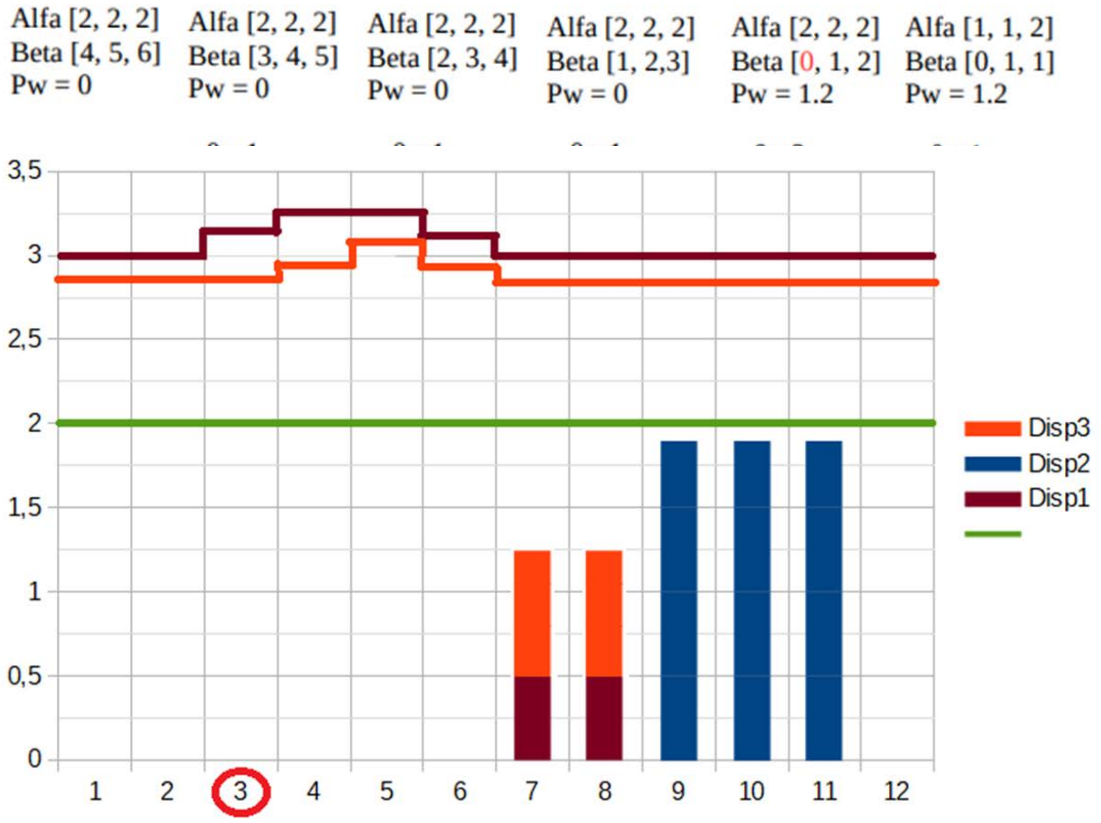
timestamp = 3 numerodispositivi = 3

soglia=[2,2,2,2,2,2,2,2,2,2,2,2]

potenzaimpegnata=np.array([[0,0,0,0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0,0,0,0]])

alfa = [2,2,2] beta = [4,5,6]

path: [((0, 0, 0), 3), ((0, 0, 0), 4), ((0, 0, 0), 5), ((0, 1, 0), 6), ((1, 1, 0), 7), ((1, 1, 0), 8), ((1, 1, 0), 9), ((1, 1, 0), 10), ((1, 1, 0), 11), ((1, 1, 1), 0), ((1, 1, 1), 1), ((1, 1, 1), 2)] con reward: 10



Anche in questo caso come per il caso di studio precedente è l'algoritmo Q_Learning a rispondere in modo corretto alla richiesta dopo 1000 iterazioni.