CityLearn

Andrea Blushi

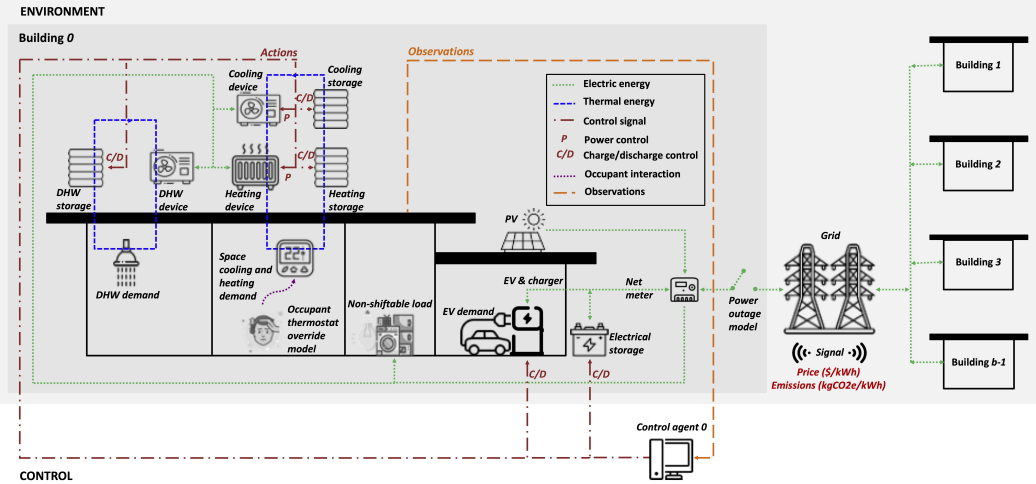
Marzo 2025

Sommario

1 Introduzione

CityLearn è un ambiente Gymnasium open source progettato per lo sviluppo e la valutazione di algoritmi, finalizzati alla gestione coordinata delle risorse energetiche a livello urbano. Il suo scopo principale è di semplificare e standardizzare il problema, permettendo di confrontare diversi algoritmi di Reinforcement Learning.

2 Ambiente di Simulazione



CityLearn è una raccolta di modelli energetici di edifici che costituiscono un distretto (chiamato anche quartiere), che include principalmente sorgenti elettriche, accumulatori di energia, batterie e dispositivi elettrici.
In ogni edificio, esistono dispositivi di riscaldamento/raffreddamento dell'ambiente, che attraverso un controllore regolano la temperatura dell'edificio in maniera ottimale. Oltre a questi, sono inclusi i dispositivi per il riscaldamento dell'acqua sanitaria. Inoltre è da precisare che ogni edificio viene modellato come una singola zona termica, e che non necessariamente tutti gli edifici implementano tutte le tipologie di dispositivi descritti. Insieme ai dispositivi, gli edifici sono dotati di una combinazione di accumulatori termici e batterie che forniscono flessibilità energetica. Questi dispositivi di accumulo possono sostituire le sorgenti di energia della rete principale nei periodi meno convenienti.
In alcuni modelli, si possono trovare gli impianti fotovoltaici, che sostituiscono completamente o parzialmente il consumo di elettricità dalla rete, generando energia in modo autonomo.
Oltre al controller principale (gestito dall'agente), è presente un controller

di backup interno ad ogni dispositivo, che fornisce dei controlli base e automatici, tra cui il dare priorità alla soddisfazione immediata dei dispositivi piuttosto che all'immagazzinamento dell'energia. Inoltre, questo controller impedisce che i dispositivi di accumulo scarichino più energia di quanto necessario per soddisfare i carichi dell'edificio. L'ambiente descrive il controller di backup attraverso una funzione opzionale *autosize()* per tutti i dispositivi ed accumulatori, che consente loro di utilizzare la potenza minima necessaria per raggiungere il carico ideale.
Altre caratteristiche specifiche delle versioni di CityLearn includono:

- La temperatura interna degli edifici può variare dinamicamente, a differenza delle versioni precedenti, in cui la temperatura era fissa. Ciò consente una gestione più efficiente dei consumi energetici.
- È possibile simulare interruzioni di corrente, durante le quali gli edifici possono utilizzare solo le risorse energetiche disponibili, come batterie e impianti fotovoltaici.
- È possibile generare dataset utilizzando il "End Use Load Profiles for U.S Buildings"(dataset che contiene informazioni dettagliate sui consumi energetici degli edifici negli Stati Uniti).
- Possono essere incluse stazioni di ricarica per veicoli elettrici, che vengono gestite molto similmente alle batterie elettriche.
- È possibile simulare il comportamento di persone che modificano manualmente la temperatura del termostato.
- È possibile tenere in considerazione oltre che al consumo energetico anche l'impatto ambientale.

2.1 Agenti di controllo

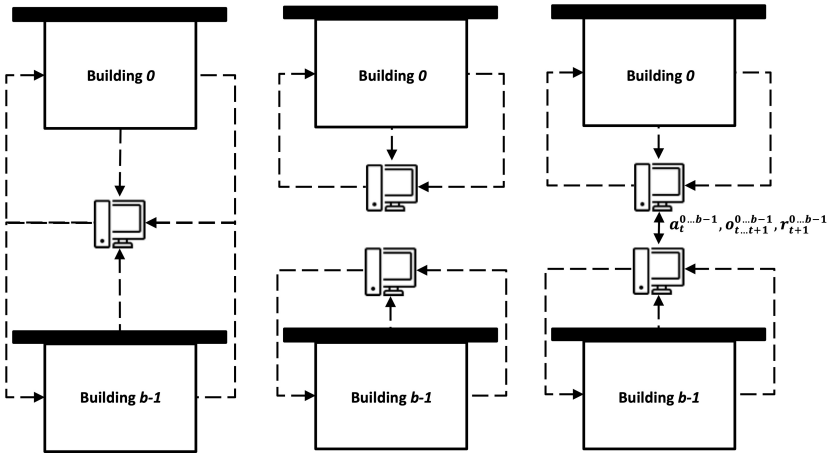
CityLearn fornisce diversi agenti di controllo, che hanno l'obiettivo di gestire i dispositivi di accumulo determinando quanto energia immagazzinare o rilasciare, oppure i dispositivi di raffreddamento e riscaldamento, stabilendo la loro potenza di fornitura ad ogni intervallo di controllo (step).

Non tutte le azioni indicate sono necessariamente incluse nello spazio delle azioni disponibili all'agente, in quanto potrebbe essere assente il dispositivo o essere abilitata la funzione *autosize* (che implica che è gestito autonomamente dall'ambiente). Inoltre, CityLearn supporta le seguenti architetture di controllo:

- **Centralizzata**:un agente controlla tutti gli accumulatori e dispositivi all'interno di un distretto.

- **Decentralizzata-indipendente**: ogni edificio ha il suo unico agente che non condivide informazioni con gli agenti di altri edifici, lavorando in maniera indipendente e isolata nel suo edificio.

- **Decentralizzata-coordinata**: simile alla decentralizzata-indipendente, con l'unica eccezione che gli agenti condivido le informazioni con lo scopo di raggiungere obiettivi comuni.



2.2 Dataset

Ogni dataset è una configurazione unica dell'ambiente che differisce in: locazione, zona climatica, caratteristiche degli edifici, serie temporali, numero di edifici, disponibilità di determinati modelli all'interno della simulazione (come dinamica termica, comportamento degli occupanti o interruzioni di corrente), variabili riguardati il prezzo e l'impatto ambientale. Tutto ciò influisce direttamente sulla complessità dei controllori. I dataset forniti da CityLearn sono composti da una raccolta di file che definiscono l'ambiente di simulazione e forniscono valori di osservazione.

I file si dividono in:

- **Building Data File**, contiene dati temporali su consumi energetici, produzione di energia solare e condizioni interne di un edificio. Ogni edificio dell'ambiente ha il proprio file.

- **Weather Data File**, contiene serie temporali su variabili meteorologiche specifiche di un luogo geografico simulato.

- **Carbon Intensity Data File**, contiene informazioni riguardanti ai tassi di emissione di CO_2, indicando l'intensità delle emissioni nel tempo.
- **Pricing Data File**, contiene dati temporali dei prezzi dell'elettricità attuali e previsti.
- **LSTM Model File**, è un file opzionale usato per simulare la variazione della temperatura all'interno degli edifici.

Tutti questi file sono in formato CSV ed i riferimenti ad essi sono contenuti e raggruppati nel file **schema.json**, che viene utilizzato per definire l'ambiente di simulazione.

3 Osservazioni

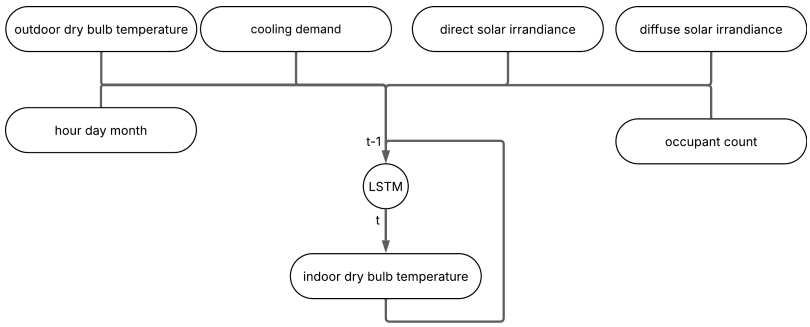
Nell'ambiente di CityLearn lo spazio degli stati coincide con le osservazioni che otteniamo ad ogni step. Queste osservazioni sono raggruppate in quattro categorie:

- **Calendar**: include variabili temporali come il mese (**month**), il giorno della settimana (**day_type**), l'ora (**hour**) e l'ora legale (**daylight_savings_status**, di tipo booleano). Tutte queste osservazioni sono dei **dati grezzi** estratti direttamente da **building.csv**.
- **Weather**: sono misure fisiche e previsioni del meteo, e sono inclusi i seguenti attributi:
  - **outdoor_dry_bulb_temperature**, misura della temperatura esterna (°C);
  - **outdoor_dry_bulb_temperature_predicted_6h, 12h, 24h**, previsioni della temperatura esterna a 6, 12 e 24 ore (°C). Si precisa che non sono array di valori, ma un singolo valore alla sesta ora ad esempio;
  - **outdoor_relative_humidity**, umidità relativa esterna (%);
  - **outdoor_relative_humidity_predicted_6h, 12h, 24h**(%);
  - **diffuse_solar_irradiance**, intensità della radiazione solare diffusa (W/m^2);
  - **diffuse_solar_irradiance_predicted_6h, 12h, 24h**;
  - **direct_solar_irradiance**, intensità della radiazione solare diretta, continua (W/m^2);

– **direct_solar_irradiance_predicted_6h, 12h, 24h**;

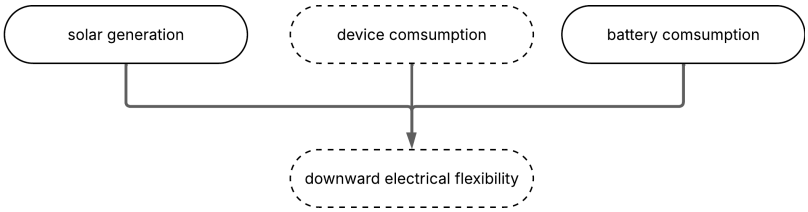
Tutti i valori sono ottenuti direttamente come **dati grezzi** da **weather.csv**

- **District**: in questa categoria sono raccolte le misurazioni che coinvolgono l'intero distretto, e sono inclusi i seguenti attributi:
  - **carbon_intensity**, indica il tasso di emissioni di CO2 per kWh complessivo di un distretto (kgCO2/kWh). Viene ottenuto direttamente come **dato grezzo** da **carbon_intensity.csv**;
  - **electricity_pricing**, prezzo corrente dell'elettricità ($/kWh) ottenuto come **dato grezzo** dal **pricing.csv**;
  - **electricity_pricing_predicted_6h, 12h, 24h**, prezzo previsto dell'elettricità per le prossime ore ($/kWh);
- **Building**: in questa categoria sono raccolti le osservazioni specifiche di ogni edificio:
  - **indoor_dry_bulb_temperature**, indica la temperatura media interna dell'edificio (°C). È un dato calcolato tramite un modello predittivo basato su **LSTM** (Long Short-Term Memory), che apprende la dinamica termica dell'edificio nel tempo. Questa temperatura è dipendente dalle azioni intraprese, e tiene conto anche di variabili come la temperatura esterna, l'irraggiamento solare e le condizioni interne, per prevedere come l'edificio reagisce termicamente nel tempo;



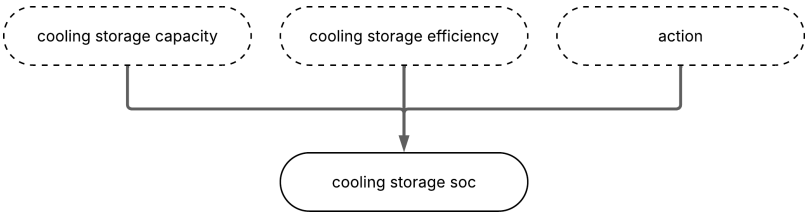
  - **average_unmet_cooling_setpoint_difference**, differenza media tra la temperatura interna e il setpoint di raffreddamento (°C), ottenuto come **dato grezzo** da **building.csv**;
  - **indoor_relative_humidity**, umidità relativa media interna all'edificio (%), ottenuto come **dato grezzo** da **building.csv**;

– **non_shiftable_load**, consumi elettrici fissi dovuti a dispositivi come lavatrici o frigoriferi (kWh). È un valore **derivato**, calcolato come il minimo tra la domanda reale dei dispositivi (disponibile nel dataset **building.csv**) e il massimo consumo elettrico ammissibile da parte dell'edificio (**downward electrical flexibility**).



– **solar_generation**, energia elettrica generata dai pannelli solari (kWh), ottenuta dal dataset **building.csv** che è fornita in input come serie temporale pre-computata;

– **cooling_storage_soc**, stato di carica dello storage di raffreddamento (0 = completamente scarico, 1 = completamente carico). É un valore **derivato** aggiornato a ogni time step sulla base delle azioni applicate e delle caratteristiche dello storage, tenendo conto anche della temperatura esterna (presa dal dataset **weather.csv**) utile per il coefficiente di efficienza. Si tiene conto della potenza disponibile nel sistema, si evita di scaricare più di quanto il sistema richieda e si calcolano i risultati effettivi della carica, applicandoli anche al dispositivo di raffreddamento. É dipendente dalle **azioni**;



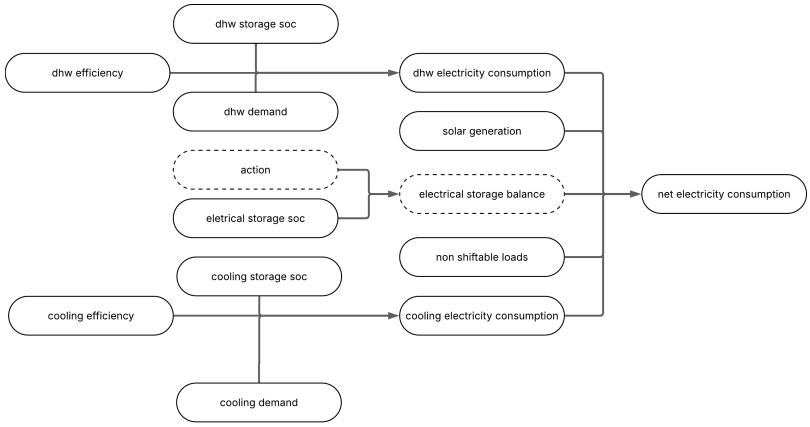
– **heating_storage_soc**, stato di carica dello storage di riscaldamento (0 scarico, 1 completamente carico), funziona analogamente al cooling_storage_soc. É dipendente dalle **azioni**;

– **dhw_storage_soc**, stato di carica dello storage per acqua calda sanitaria (0 scarico, 1 completamente carico), funziona analogamente al cooling_storage_soc. É dipendente dalle **azioni**;

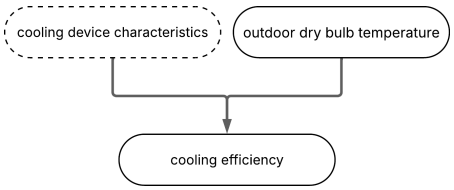
– **electrical_storage_soc**, stato di carica dello storage elettrico (0 scarico, 1 completamente carico). Un valore **derivato** ottenuto direttamente dall'energia elettrica caricata o scaricata (influenzato dalle azioni) dalla batteria elettrica, limitata dalla capacità

dell'accumulatore e dalla flessibilità elettrica disponibile nel time step. É dipendente dalle **azioni**;

– **net_electricity_consumption**, consumo netto totale di elettricità (kWh), ottenuto dal calcolo **derivato** dei consumi di tutti i dispositivi e batterie, contrastato dal pannello fotovoltaico. Vengono ignorati i consumi degli accumulatori termici in quanto vengono caricati dai corrispettivi dispositivi, dei quali si ha già registrato il consumo.



– **cooling_device_efficiency**, efficienza del dispositivo di raffreddamento, che corrisponde al coefficiente di efficienza il quale si ottiene dal calcolo **derivato** da temperatura esterna (ottenuta da **weather.csv**) e dalle caratteristiche del dispositivo predefinite;

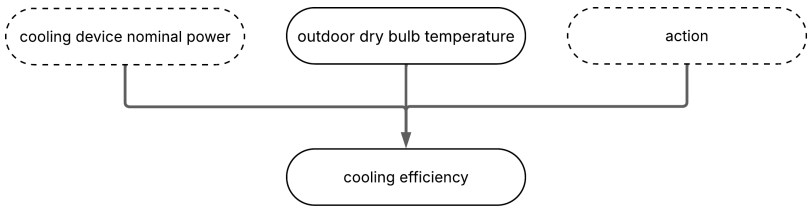


– **heating_device_efficiency**, efficienza del dispositivo di riscaldamento, ottenuto analogamente al dispositivo di raffreddamento;

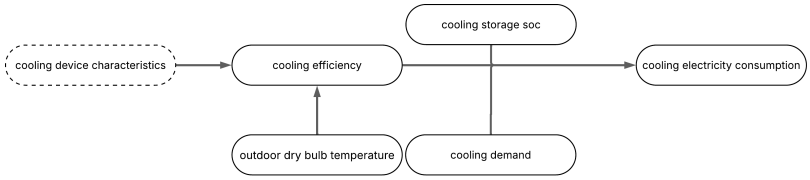
– **dhw_device_efficiency**, efficienza del dispositivo per l'acqua calda sanitaria, ottenuto analogamente al dispositivo di raffreddamento;

– **cooling_demand**, domanda energetica per il raffreddamento (kWh). Rappresenta la quantità di energia termica utilizzata per mantenere la temperatura interna dell'edificio entro i limiti di comfort durante le ore calde. Il dato **derivato** viene calcolato come

la somma dell'energia termica fornita direttamente dal dispositivo di raffreddamento e dell'energia prelevata dall'accumulatore di raffreddamento. É dipendente dalle **azioni**;



- **heating_demand**, domanda energetica per il riscaldamento (kWh), ottenuto analogamente al dispositivo di raffreddamento. É dipendente dalle **azioni**;
- **dhw_demand**, domanda energetica per l'acqua calda sanitaria (kWh), ottenuto analogamente al dispositivo di raffreddamento. É dipendente dalle **azioni**;
- **cooling_electricity_consumption**, consumo elettrico per soddisfare la domanda di raffreddamento (kWh). È un dato **derivato** che viene calcolato come il rapporto tra la domanda di raffreddamento erogata e il coefficiente di performance del dispositivo. É dipendente dalle **azioni**;



- **heating_electricity_consumption**, consumo elettrico per soddisfare la domanda di riscaldamento (kWh), ottenuto analogamente al dispositivo di raffreddamento. É dipendente dalle **azioni**;
- **dhw_electricity_consumption**, consumo elettrico per l'acqua calda sanitaria (kWh), ottenuto analogamente al dispositivo di raffreddamento. É dipendente dalle **azioni**;
- **indoor_dry_bulb_temperature_set_point**, valore del setpoint della temperatura interna (°C), ottenuto come **dato grezzo** dal file **building.csv**, oppure modificato dinamicamente in presenza di occupanti tramite **modelli predittivi e probabilistici** (Logistic Regression Occupant);

- **indoor_dry_bulb_temperature_delta**, differenza assoluta tra la temperatura interna e il relativo setpoint (°C), ottenuto come valore **derivato**;
- **occupant_count**, numero di occupanti presenti nell'edificio al step corrente, fornito come **dato grezzo** dal dataset **building.csv**;
- **comfort_band**, comfort termico accettabile per l'edificio (°C), fornito come **dato grezzo** dal dataset **building.csv**;
- **power_outage**, indica se c'è un'interruzione di corrente elettrica (0 = no, 1 = sì), fornito come **dato casuale** da un modello stocastico basato su metriche di affidabilità realistiche come SAIFI (frequenza media di interruzioni) e CAIDI (durata media delle interruzioni);
- **hvac_mode**, modalità operativa corrente del sistema di climatizzazione, fornito come **dato grezzo** dal dataset **building.csv**;

Lo spazio degli stati è principalmente **continuo**, in quanto costituito per lo più da variabili continue, con rare eccezioni come power_outage.

Di seguito si trovano gli schemi che riassumo i comportamenti delle osservazioni e come interagiscono fra di loro:

4 Spazio delle azioni

Nell'ambiente di CityLearn lo spazio delle azioni è definito dalle **azioni di controllo** che possono essere applicate ai vari dispositivi dell'edificio, come accumulatori o dispositivi di climatizzazione. Le azioni disponibili sono:

- **cooling_storage** $[-1, 1]$: proporzione della capacità del sistema di accumulo del raffreddamento da caricare ($a > 0$) o scaricare ($a < 0$).
- **heating_storage** $[-1, 1]$: proporzione della capacità del sistema di accumulo del riscaldamento da caricare ($a > 0$) o scaricare ($a < 0$).
- **dhw_storage** $[-1, 1]$: proporzione della capacità del sistema di accumulo dell'acqua calda sanitaria da caricare ($a > 0$) o scaricare ($a < 0$).
- **electrical_storage** $[-1, 1]$: proporzione della capacità della batteria da caricare ($a > 0$) o scaricare ($a < 0$);
- **electric_vehicle_storage** $[-1, 1]$: proporzione della capacità di carica del veicolo elettrico da caricare ($a > 0$) o scaricare ($a < 0$).

- **cooling_device** [0, 1]: proporzione della potenza nominale da erogare per il dispositivo di raffreddamento.
- **heating_device** [0, 1]: proporzione della potenza nominale da erogare per il dispositivo di riscaldamento.
- **cooling_or_heating_device** [−1, 1]: proporzione della potenza nominale da erogare per il dispositivo ibrido, per ($a > 0$) riscalda l'ambiente e per ($a < 0$) lo raffredda. (Non è dichiarato esplicitamente nella documentazione ma considerato nel codice)

È importante sottolineare che lo spazio delle azioni è **continuo**, in quanto ogni valore di controllo rientra in un intervallo reale. Questa caratteristica consente di impostare con precisione l'intensità di carica/scarica degli accumulatori o la potenza dei dispositivi di climatizzazione, non limitandosi a semplici on/off. Inoltre, in uno stesso edificio possono non essere presenti tutti i dispositivi oppure alcuni di essi possono essere impostati in *autosize*, determinando la loro assenza nello spazio delle azioni.

Dal punto di vista implementativo, il vettore delle azioni può variare forma a seconda che l'agente sia **centrale** o **distribuito**.

- **Agente centrale**: tutti i valori di azione vengono concatenati in un singolo vettore, che viene poi suddiviso per ogni edificio.

```
[[dhw_storage, electrical_storage, cooling_device,
 dhw_storage, electrical_storage, cooling_device,
dhw_storage, electrical_storage, cooling_device]]
```

- **Agente distribuito**: ogni edificio riceve direttamente un proprio vettore di azioni.

```
[[dhw_storage, electrical_storage, cooling_device],
 [dhw_storage, electrical_storage, cooling_device],
 [dhw_storage, electrical_storage, cooling_device]]
```

5 Modello di transizione

Nel contesto di CityLearn, il modello di transizione è determinato dalle dinamiche fisiche e operative che possono essere simulate all'interno dell'ambiente. Tra queste sono incluse:

- **Dinamiche termiche**: raccogliendo dati come condizioni meteorologiche esterne e il funzionamento dei dispositivi di climatizzazione, restituisce variazioni della temperatura ed altri valori che descrivono le dinamiche termiche di un edificio. Questo comportamento viene descritto nel file *dynamics.py*.

- **Comportamento dei dispositivi ed accumulatori**: date le azioni dell'agente ed osservazioni dell'edificio, CityLearn fornisce un modello di comportamento per ogni dispositivo, descrivendone consumi ed efficacia. Tutti i modelli dei dispositivi sono contenuti in *energy_model.py*. Inoltre, se inclusi nell'ambiente, i veicoli elettrici sono considerabili come dispositivi, ma il loro comportamento è descritto in *electric_vehicle.py*

- **Comportamento degli occupanti**: descrive in maniera casuale o stocastica i comportamenti degli occupanti o il loro numero all'interno dell'edifico. L'azione principale che può svolgere un occupante è la modifica della temperatura di setpoint. Il modello è descritto in *occupant.py*

- **Comportamenti della rete elettrica**: l'ambiente raccoglie tutti i dati dai modelli descritti precedentemente, restituendo valori complessivi di tutta la rete e descrivendo in questo modo il comportamento della rete. Risulta esclusa la casistica dell'interruzione elettrica, che viene modellata come casuale e stocastica nel file *power_outage.py*.

Il modello di transizione in CityLearn è prevalentemente **deterministico**: se ripetute le stesse azioni in condizioni identiche vengono prodotti gli stessi risultati. Questo perché le osservazioni, come le condizioni meteorologiche, sono predefinite nei dataset. Tuttavia, esistono delle eccezioni, come il comportamento degli occupanti o le interruzioni di corrente, che vengono modellati come eventi casuali. Tuttavia, impostando un seed nell'ambiente, anche questi elementi tornano a essere deterministici.

6 Reward Function

Ad ogni *step* della simulazione viene calcolato un **reward** attraverso la chiamata al modulo *citylearn.citylearn.CityLearnEnv.reward_functions*. Il modulo offre una serie di funzioni di base, pur dando possibilità all'utente di aggiungere funzioni personalizzate. È importante sottolineare che, in CityLearn, la funzione di reward viene definita in modo statico all'interno

del file di configurazione *schema.json*. Questo significa che la funzione di ricompensa è specificata una volta all'inizio e rimane invariata durante l'intero processo di simulazione, a meno che non venga modificata manualmente.

6.0.1 Base Reward Function

$$\mathbf{min}(-\boldsymbol{e}, \mathbf{0})$$

Calcola il minimo tra $-\boldsymbol{e}$ e $\mathbf{0}$, dove $\boldsymbol{e}$ è il consumo netto di elettricità di un edificio. Questa funzione viene utilizzata nell'algoritmo di baseline ed il suo obiettivo è di penalizzare i consumi di elettricità.

6.0.2 MARL Reward

$$\mathbf{sign}(-\boldsymbol{e}) \times \mathbf{0.01}(\boldsymbol{e^2}) \times \mathbf{max}(\mathbf{0}, \boldsymbol{E})$$

Combina il segno di $-\boldsymbol{e}$ con una funzione quadratica di $\boldsymbol{e}$ e il massimo tra $\mathbf{0}$ e $\boldsymbol{E}$. Si analizzano le varie componenti di seguito:

- $\mathbf{sign}(-\boldsymbol{e})$: se $-\boldsymbol{e} > \mathbf{0}$ allora significa che l'edificio consuma più di quello che produce, ed il segno porta a penalizzare questo comportamento. Se $-\boldsymbol{e} < \mathbf{0}$ allora significa che l'edificio consuma meno di quello che produce, ed il segno tende a premiare questo comportamento.
- $\mathbf{0.01}(\boldsymbol{e^2})$: in questo caso maggiore sarà il consumo, maggiore sarà la penalità (il fattore quadratico), portando così il sistema a penalizzare i picchi eccessivi di consumo. Il fattore $\mathbf{0.01}$ ha lo scopo di ridurre l'impatto della penalità.
- $\boldsymbol{max}(\mathbf{0}, \boldsymbol{E})$: $\boldsymbol{E}$ rappresenta il consumo complessivo del distretto. In questo caso lo scopo è di allineare tutti gli edifici ad un obiettivo comune portando a diminuire così il consumo del distretto.

6.0.3 Independent SAC Reward

$$\mathbf{min}(-\boldsymbol{e^3}, \mathbf{0})$$

Come nella funzione di reward di base, ha l'obiettivo di penalizzare i picchi di consumo del sistema, enfatizzando ciò con il fattore cubico.

6.0.4 Solar Penalty Reward

$$\sum_{i=0}^{n} -(\left(\mathbf{1} + \frac{\boldsymbol{e}}{|\boldsymbol{e}|} \times \mathbf{storage}^{\boldsymbol{SoC}}\right) \times |\boldsymbol{e}|)$$

Il Solar Penalty Reward è progettato per incentivare comportamenti che minimizzano il consumo di energia elettrica dalla rete e massimizzano l'uso dell'energia solare. Si analizzano le varie componenti di seguito:

- $\frac{\boldsymbol{e}}{|\boldsymbol{e}|}$: serve a distinguere il consumo netto ($\boldsymbol{e} > \mathbf{0}$) e produzione netta ($\boldsymbol{e} < \mathbf{0}$) ritornando **1** o **-1** a seconda dei casi.
- $\mathbf{storage}^{\boldsymbol{SoC}}$: rappresenta lo stato di carica del sistema di accumulo di energia, ed in questo caso serve per modulare il reward in base alla capacità di accumulo disponibile.
- $\left(\mathbf{1} + \frac{\boldsymbol{e}}{|\boldsymbol{e}|} \times \mathbf{storage}^{\boldsymbol{SoC}}\right)$: serve ad aumentare la penalità quando l'edificio consuma energia e il sistema di accumulo non è completamente carico, oppure quando l'edificio produce energia in rete e il sistema di accumulo non è completamente scarico.
- $|\boldsymbol{e}|$: prende il valore del consumo netto per garantire che la penalità sia proporzionale.

La sommatoria scorre tutte le batterie disponibili nell'ambiente, andando a verificare così il loro stato.

6.0.5 ComfortReward

$$\begin{cases} -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}|^{\mathbf{3}} & \text{if } \boldsymbol{T_{in}} < (\boldsymbol{T_{stp}} - \boldsymbol{T_b}) \text{ and cooling} \\ -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}|^{\mathbf{2}} & \text{if } \boldsymbol{T_{in}} < (\boldsymbol{T_{stp}} - \boldsymbol{T_b}) \text{ and heating} \\ -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}| & \text{if } (\boldsymbol{T_{stp}} - \boldsymbol{T_b}) \leq \boldsymbol{T_{in}} < \boldsymbol{T_{stp}} \text{ and cooling} \\ \mathbf{0} & \text{if } (\boldsymbol{T_{stp}} - \boldsymbol{T_b}) \leq \boldsymbol{T_{in}} < \boldsymbol{T_{stp}} \text{ and heating} \\ \mathbf{0} & \text{if } \boldsymbol{T_{stp}} \leq \boldsymbol{T_{in}} \leq (\boldsymbol{T_{stp}} + \boldsymbol{T_b}) \text{ and cooling} \\ -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}| & \text{if } \boldsymbol{T_{stp}} \leq \boldsymbol{T_{in}} \leq (\boldsymbol{T_{stp}} + \boldsymbol{T_b}) \text{ and heating} \\ -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}|^{\mathbf{2}} & \text{if } (\boldsymbol{T_{stp}} + \boldsymbol{T_b}) < \boldsymbol{T_{in}} \text{ and cooling} \\ -|\boldsymbol{T_{in}} - \boldsymbol{T_{stp}}|^{\mathbf{3}} & \text{otherwise} \end{cases}$$

Funzione complessa che calcola un reward basato sulla temperatura interna $\boldsymbol{T_{in}}$, la temperatura di setpoint $\boldsymbol{T_{stp}}$, e la temperatura di comfort $\boldsymbol{T_b}$, considerando sia i flag di **raffreddamento** e **riscaldamento**. Il suo scopo è di incentivare il mantenimento della temperatura interna, penalizzare gli scostamenti eccessivi ed ottimizzare il comfort termico.

6.0.6 V2G Penalty Reward

La classe V2GPenaltyReward calcola penalità e ricompense relative al comportamento di ricarica dei veicoli elettrici in un contesto Vehicle-to-Grid (V2G), con l'obiettivo di promuovere un uso efficiente dell'energia. A livello implementativo, funziona come un plugin aggiuntivo che si integra con altri reward, estendendo la funzione di ricompensa principale per includere anche l'utilizzo e la gestione dei veicoli elettrici. Quindi non va utilizzato autonomamente. La formula seguente evidenzia ciò, infatti il reward risultate è la somma di un reward base più il V2GPenaltyReward.

$$\boldsymbol{R} = \boldsymbol{R}_{\text{base}} + \boldsymbol{P}_{\text{EV}}$$

Nello specifico $\boldsymbol{P}_{\text{EV}}$ è descrivibile in questo modo:

$$\boldsymbol{P}_{\text{EV}} = \boldsymbol{P}_{\text{NoCar}} + \boldsymbol{P}_{\text{BatteryLimits}} + \boldsymbol{P}_{\text{SoCDiff}} + \boldsymbol{P}_{\text{SelfProd}} + \boldsymbol{P}_{\text{SelfEVCons}}$$

I termini sono definiti come:

- $\boldsymbol{P}_{\text{NoCar}}$: penalità per ricarica senza auto presente.
- $\boldsymbol{P}_{\text{BatteryLimits}}$: penalità per superamento dei limiti della batteria.
- $\boldsymbol{P}_{\text{SoCDiff}}$: penalità o ricompensa per la differenza tra stato della batteria attuale e richiesto.
- $\boldsymbol{P}_{\text{SelfProd}}$: ricompensa per autoconsumo o produzione extra.
- $\boldsymbol{P}_{\text{SelfEVCons}}$: ricompensa per l'uso del veicolo elettrico per supportare l'edificio.

6.0.7 Solar Penalty And Comfort Reward

Questo reward implementa direttamente la combinazione del Solar Penalty Reward e del Comfort Reward descritti in precedenza, unificandoli in un'unica formula a cui vengono assegnati, di default, pesi uguali. In questo modo, viene data priorità a entrambi gli aspetti in modo bilanciato.

$$\text{Reward}_{\text{totale}} = \boldsymbol{\alpha} \cdot \text{Reward}_{\text{solar}} + \boldsymbol{\beta} \cdot \text{Reward}_{\text{comfort}}$$

6.1 Custom Reward

Oltre ai reward standard già presenti, prenderemo in considerazione anche altri reward personalizzati, creati direttamente da noi in funzione degli obiettivi specifici del nostro sistema. Inoltre, includeremo alcuni reward ottenuti dalle CityLearn Challenge 2023, in quanto si tratta dell'unica edizione che introduce una penalizzazione per il discomfort dell'utente.

6.1.1 Comfort and Consumption Reduction Reward

Il Comfort and Consumption Reduction Reward è una funzione di reward ispirata al modello ideato al migliore della CityLearn Post Challenge 2023. Essa estende il classico Comfort Reward Function aggiungendo penalità mirate sui consumi energetici, con l'obiettivo di bilanciare comfort termico, stabilità nei consumi e resilienza ai blackout. Di seguito la formula che riassume la logica:

$$\boldsymbol{R = -0.3 \cdot |\Delta T| - 0.1 \cdot s - 0.15 \cdot |r|}$$

dove:

- $\boldsymbol{\Delta T} = \begin{cases} \boldsymbol{3 \cdot (T_{\text{indoor}} - T_{\text{setpoint}})}, & \text{se } \boldsymbol{T_{\text{indoor}} < T_{\text{setpoint}}} \\ \boldsymbol{T_{\text{indoor}} - T_{\text{setpoint}}}, & \text{altrimenti} \end{cases}$

  Rappresenta la deviazione della temperatura interna rispetto al setpoint. Le deviazioni verso una temperatura troppo fredda sono penalizzate maggiormente, riflettendo un discomfort più severo. Da tener conto che questo reward è specifico per periodi estivi, penalizzando l'eccessivo raffreddamento che consegue ad eccessivo consumo inutile.

- $\boldsymbol{r = C_t - C_{t-1}}$

  E' il ramping, ovvero la variazione del consumo elettrico netto tra due timestep consecutivi. Questo termine penalizza cambiamenti bruschi nel profilo di consumo, incentivando un comportamento più stabile.

- $\boldsymbol{s} = \begin{cases} \boldsymbol{L_{\text{non-shiftable}} - C_t}, & \text{se power\_outage} = \boldsymbol{1} \\ \boldsymbol{0}, & \text{altrimenti} \end{cases}$

  Rappresenta la quota di carico essenziale non soddisfatto durante un blackout. Questo termine incentiva la continuità della fornitura energetica per i carichi non gestiti (come frigoriferi).

Inoltre lo sviluppatore fornisce porzioni di codice commentate, che suggeriscono un'implementazione per la gestione di emissioni di CO2. Queste si aggiungono con la seguente formula:

$$-0.05 \cdot (\max(0, C \cdot \gamma) - 7000)$$

C = net electricity consumption $\quad \gamma$ = carbon emission rate

7 Cost Function

CityLearn fornisce un insieme di funzioni che fungono da indicatori per valutare la situazione energetica di distretti ed edifici. La libreria include la funzione **evaluate()**, utile per il confronto con l'algoritmo di *baseline*, che restituisce una tabella (divisa per ogni edificio) contenente i seguenti parametri:
(Tutti i parametri sono normalizzati e rapportati all'algoritmo di baseline)

- **all_time_peak_average**: rapporto tra i picchi massimi di consumo della rete.
- **annual_normalized_unserved_energy_total**: proporzione della domanda non soddisfatta a causa di carenza di fornitura.
- **carbon_emissions_total**: emissioni totali di carbonio.
- **cost_total**: costo totale dell'elettricità.
- **daily_one_minus_load_factor_average**: rapporto medio tra il consumo medio giornaliero e il picco di consumo. Il fattore di carico (load factor) rappresenta l'efficienza del consumo di elettricità ed è compreso tra 0 (molto inefficiente) e 1 (altamente efficiente). Indica sprechi di energia.
- **daily_peak_average**: picco medio giornaliero di consumo elettrico.
- **discomfort_cold_delta_average**: deviazione media della temperatura al di sotto del setpoint di riscaldamento.
- **discomfort_cold_delta_maximum**: deviazione massima registrata per eccessivo freddo percepito.
- **discomfort_cold_delta_minimum**: deviazione minima del discomfort da freddo.

- **discomfort_cold_proportion**: proporzione di tempo in cui si verifica discomfort da freddo.
- **discomfort_hot**: parametri equivalenti al discomfort da freddo, ma relativi al caldo.
- **discomfort_proportion**: percentuale totale del tempo in cui l'utente si trova fuori dalla fascia di comfort.
- **electricity_consumption_total**: consumo totale di elettricità, normalizzato rispetto allo scenario baseline.
- **monthly_one_minus_load_factor_average**: utile per analisi a lungo termine dell'efficienza energetica.
- **one_minus_thermal_resilience_proportion**: proporzione di discomfort durante un blackout.
- **power_outage_normalized_unserved_energy_total**: energia non fornita durante un blackout.
- **ramping_average**: media della variazione del consumo elettrico netto tra due timestep consecutivi. Indica la regolarità del profilo di consumo del distretto: un valore basso di ramping indica un cambiamento graduale del consumo, anche dopo che l'auto-produzione non è più disponibile la sera e al mattino presto. Un valore alto di ramping indica invece un cambiamento brusco del carico sulla rete, che può causare stress non pianificato sull'infrastruttura.
- **zero_net_energy**: misura la vicinanza al bilancio nullo tra produzione e consumo energetico.

8 Algoritmi di Controllo

CityLearn fornisce l'implementazione di algoritmi **Rule-based control** (RBC) e **Reinforcement learning control** (RLC).
Gli algoritmi RBC, disponibili nel modulo *citylearn.agents.rbc*, si basano su un'architettura centralizzata e indipendente. Invece gli algoritmi RLC supportano sia un'architettura centralizzata e indipendente (disponibile nel modulo *citylearn.agents.sac*) sia un'architettura decentralizzata e coordinata (disponibile nel modulo *citylearn.agents.marlisa*). Oltre agli algoritmi RBC e RLC, CityLearn include anche agenti di riferimento utili per stabilire basi di confronto e poter valutare gli altri agenti. CityLearn, nel modulo *base.py*,

fornisce una struttura da cui partire per la creazione di nuovi algoritmi personalizzati per l'utente.
Di seguito vengono approfonditi tutti gli algoritmi, mettendo a confronto le loro caratteristiche attraverso alcune *cost_function* della simulazione *citylearn_challenge_2023*

8.1 Baseline Agent

Implementazione Questo agente viene utilizzato per simulazioni di base, senza alcun tipo di controllo sugli accumulatori o sulle pompe di calore. L'ambiente, con questo agente, viene modificato disattivando tutte le azioni, impostando così tutti i dispositivi in una situazione simil **autosize**, fornendo i carichi ideali nell'ambiente. Lo scopo principale di questo agente è di fungere da riferimento per confrontare l'efficienza e i parametri degli altri agenti attraverso le *cost_function*.

Valutazione

name	Building_1	Building_2	Building_3	District
cost_function				
carbon_emissions_total	1.000	1.000	1.000	1.000
discomfort_proportion	0.377	0.041	0.098	0.172
electricity_consumption_total	1.000	1.000	1.000	1.000

8.2 Rule-Based Control - RBC

Gli algoritmi RBC seguono regole basate su soglie statiche predefinite per gestire il consumo e lo stoccaggio di energia, limitando però la capacità di adattamento alle condizioni variabili dell'ambiente. Di conseguenza, gli algoritmi RBC risultano semplici, presentando però una mancanza di ottimizzazione in scenari complessi.

8.2.1 HourRBC

Implementazione HourRBC, disponibile nel modulo *citylearn.agents.rbc*, funge da interfaccia base, offrendo la struttura su cui si basano gli altri agenti RBC. A differenza degli altri algoritmi, non implementa direttamente regole specifiche, ma si occupa di fornire una mappatura tra le ore della giornata e le azioni corrispondenti, permettendo così una gestione temporale delle decisioni. Questo approccio consente agli agenti di seguire un principio di regolazione basato su fasce orarie, rendendo il controllo più strutturato e facilmente adattabile alle esigenze del sistema.

8.2.2 BasicRBC

Implementazione La classe BasicRBC è un agente di controllo che opera su regole orarie per sistemi di accumulo di energia e pompe di calore, costruite dagli sviluppatori basandosi sul COP (Coefficent of Perfomance). Il COP è un indice di efficienza per le pompe di calore che indica quanta energia viene prodotta rispetto all'energia che viene consumata. Ad esempio, un COP di 4 significa che la pompa di calore genera 4 unità di calore per ogni unità di elettricità usata.
Come anche gli algoritmi seguenti, vengono utilizzate le osservazioni esclusivamente per comprendere l'orario, ignorando completamente parametri come temperatura o umidità, rendendo tutti gli algoritmi RBC estremamente rigidi. Inoltre, gli accumulatori, che siano di raffreddamento/riscaldamento oppure batterie, vengono considerati uguali dall'algoritmo.
A livello pratico, per quanto riguarda il BasicRBC, le regole sono progettate in modo che l'agente carichi gli accumulatori del 9.1% della sua capacità massima tra le 22:00 e le 8:00, e scarichi l'8% della sua capacità massima nell'orario restante. I dispositivi di raffreddamento sono impostati al 40% della potenza nominale (0.4 nello spazio delle azioni) tra le 22:00 e le 8:00, e dell'80% in ogni altra ora. I dispositivi di riscaldamento, se attivi, risultano inversi a quelli di raffreddamento.

Valutazione

name	Building_1	Building_2	Building_3	District
cost_function				
carbon_emissions_total	1.998	1.936	1.740	1.891
discomfort_proportion	0.975	0.907	0.957	0.946
electricity_consumption_total	1.995	1.963	1.755	1.904

Come si può notare, i consumi e le emissioni risultano quasi il doppio rispetto all'algoritmo di baseline. Come anche il discomfort_propotion che risulta estremamente elevato. Unica nota positiva è l'energia consumata dal dispositivo di raffreddamento, in quanto risulta utilizzata in maniera più efficiente grazie alla progettazione basata sul COP.
L'inefficienza di tale algoritmo, come detto anche precedentemente, è dovuta alla rigidità che ignora completamente la situazione reale dell'ambiente, e fornisce una soluzione basata su stime e calcoli precedenti.

8.2.3 OptimizedRBC

Implementazione E' una versione ottimizzata del BasicRBC, dove le azioni di controllo sono state selezionate tramite una ricerca su griglia. Il termine "search grid" (ricerca su griglia) fa riferimento ad una tecnica utilizzata per ottimizzare parametri in modelli o algoritmi. In questo contesto implica l'esplorazione di diverse combinazioni di parametri di controllo per poi trovare la combinazione migliore (prima della progettazione dell'algoritmo). Infatti, si hanno delle regole più capillari e precise rispetto al BasicRBC.
Nello specifico, le azioni sono progettate in modo che l'agente scarichi il sistema di accumulo controllato del 2% ogni ora tra le 07:00 e le 15:00, scarichi il 4,4% ogni ora tra le 16:00 e le 18:00, scarichi il 2,4% ogni ora tra le 19:00 e le 22:00, carichi il 3,4% tra le 23:00 e mezzanotte e carichi il 5,532% ogni altra ora. Il dispositivo di raffreddamento rende disponibile il 70,0% della sua potenza nominale ogni ora tra le 07:00 e le 15:00, il 60,0% tra le 16:00 e le 18:00, l'80,0% tra le 19:00 e le 22:00, il 40,0% tra le 23:00 e mezzanotte e il 20,0% in ogni altra ora. I dispositivi di riscaldamento risultano inversi a quelli di raffreddamento.

Valutazione

name	Building_1	Building_2	Building_3	District
cost_function				
carbon_emissions_total	1.827	1.779	1.612	1.739
discomfort_proportion	0.975	0.858	0.932	0.922
electricity_consumption_total	1.811	1.788	1.615	1.738

Nonostante il tentativo di ottimizzazione, il miglioramento delle prestazioni risulta estremamente lieve, riportandoci alle stesse problematiche che affliggono il BasicRBC.

8.2.4 BasicBatteryRBC

Implementazione Un agente di controllo basato su HourRBC, progettato per sfruttare al meglio l'energia solare per la ricarica delle batterie. Durante le ore generalmente più soleggiate della giornata, l'edificio entra in una fase di ricarica, per poi rilasciare l'energia accumulata al termine di questo periodo, riducendo così il consumo di elettricità prelevata dalla rete. Inoltre, questo algoritmo tende a utilizzare i sistemi di climatizzazione in misura minore rispetto agli altri algoritmi.
Nello specifico, le azioni sono ottimizzate per la batteria in modo che l'agente

carichi dell'11,0% della sua capacità massima ogni ora tra le 06:00 e le 14:00, e scarichi il 6,7% della sua capacità massima nelle ore restanti. Il dispositivo di raffreddamento è impostato al 70,0% della potenza nominale tra le 06:00 e le 14:00 e al 30,0% in ogni altra ora. I dispositivi di riscaldamento risultano inversi a quelli di raffreddamento.

Valutazione

name	Building_1	Building_2	Building_3	District
cost_function				
carbon_emissions_total	1.506	1.493	1.377	1.459
discomfort_proportion	0.940	0.604	0.586	0.710
electricity_consumption_total	1.511	1.514	1.392	1.472

L'utilizzo più efficiente della batteria solare comporta un miglioramento significativo nelle performance di consumo energetico. Tuttavia, il miglioramento del comfort non è direttamente attribuibile a una gestione più efficiente dell'energia, ma piuttosto al fatto che la minore attivazione dei dispositivi consente all'edificio di mantenere la temperatura sopra il setpoint. Al contrario, negli altri algoritmi, l'uso eccessivo dei dispositivi di climatizzazione tende a portare la temperatura ben al di sotto del valore desiderato, uscendo facilmente dalla banda di comfort. Ciò potrebbe tranquillamente cambiare in dataset in cui si raggiungono temperature esterne più elevate, portando alla necessità di utilizzare di più i dispositivi di raffreddamento. Infine, nonostante l'utilizzo ridotto del dispositivo di raffreddamento, sembra consumare lo stesso quantitativo di energia degli altri algoritmi, in quanto il COP non è obiettivo principale di tale algoritmo.

8.3 Reinforcement Learning Control - RL

Un agente RLC interagisce con l'ambiente, ottenendo ottimalità attraverso tentativi ed errori che restituiscono ricompense basate sull'efficacia delle sue azioni. A differenza degli algoritmi RBC, vi è una notevole capacità di adattarsi a più scenari, pagando però in termini di efficienza e semplicità dell'algoritmo. CityLearn supporta i seguenti approcci RLC.

8.3.1 Tabular Q-Learning

Il Q-Learning è un algoritmo di Reinforcement Learning (RL) di tipo model-free, ovvero non richiede una conoscenza esplicita del modello dell'ambiente.

Nonostante ciò, consente a un agente di apprendere una politica ottimale interagendo direttamente con l'ambiente ed elaborando le esperienze raccolte.

Concetti teorici In problemi semplici composti con stati finiti ed azioni discrete, è possibile rappresentare tutte le transizioni stato-azione (Q-values) attraverso le tabelle, portandoci così alla variante **Tabular Q-Learning**. A livello pratico l'apprendimento viene effettuato calcolando l'**equazione di Bellman** e memorizzando il risultato sulla tabella.
Si descrive di seguito l'equazione di Bellman:

$$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]}$$

- $\boldsymbol{a}$: azione
- $\boldsymbol{s}$: stato
- $\boldsymbol{Q(s,a)}$: il valore corrispondente nella tabellla che rappresenta la coppia stato-azione;
- $\boldsymbol{r}$: è la ricompensa ottenuto per aver eseguito l'azione $\boldsymbol{a}$ nello stato $\boldsymbol{s}$;
- $\boldsymbol{\alpha}$: è il tasso di apprendimento che va da 0 a 1 e determina quanto l'agente deve aggiornare il Q-value ad ogni iterazione;
- $\boldsymbol{\gamma}$: è il fattore di sconto che determina quanto l'agente deve considerare le ricompense future rispetto a quelle immediate;
- $\boldsymbol{\max_{a'} Q(s',a')}$: permette di massimizzare la ricompensa cumulativa a lungo termine, con lo scopo di trovare la policy ottimale.

In Q-Learning, l'agente sceglie le azioni attraverso un bilanciamento tra l'esplorazione (*exploration*) di nuove e lo sfruttamento (*exploitation*) delle azioni che hanno già dato una buona ricompensa. L'implementazione in CityLearn utilizza l'approccio **epsilon-greedy** per questo bilanciamento. Infatti seleziona un'azione casuale con probabilità ϵ (per l'esplorazione) e l'azione con la ricompensa più alta con probabilità $1 - \epsilon$, aiutando l'agente ad evitare di fermarsi ai minimi locali e cercare l'azione ottimale.
Infine, la policy ottimale π si ottiene massimizzando i Q-values, arrivando dopo una serie di tentativi ad un punto di convergenza.

Di seguito di trova l'algoritmo generale:

1. Inizializzare la Q-table per tutte le coppie stato-azione.

2. Impostare il tasso di apprendimento α ($0 < \alpha < 1$) e il fattore di sconto γ ($0 < \gamma < 1$).

3. Ripetere i seguenti passi per ogni episodio:

   - Osservare lo stato iniziale s.
   - Scegliere un'azione a in base alla politica epsilon-greedy (un'azione casuale viene scelta con probabilità ϵ, mentre l'azione con il Q-value più alto viene scelta con probabilità $1 - \epsilon$).
   - Eseguire l'azione a e osservare il successivo stato s' e la ricompensa r.
   - Aggiornare il Q-value della coppia stato-azione (s, a) utilizzando l'equazione di Bellman.
   - Impostare $s = s'$.

4. Ripetere il passo 3 per un numero elevato di episodi o fino al raggiungimento della convergenza.

Implementazione Nell'ambiente CityLearn, le osservazioni e le azioni sono continue, risultando problematico l'algoritmo di Q-Learning, che è progettato per spazi discreti. Tuttavia, il modulo *citylearn.citylearn.wrappers.py* consente di discretizzare l'ambiente, grazie all'uso delle librerie di *gymnasium*, rendendo possibile applicare il Tabular Q-Learning. Attraverso l'utilizzo dei **wrapper** di gymnasium, è possibile suddividere gli intervalli di valori continui in sotto-intervalli discreti, assegnando a ciascun sottointervallo un valore discreto. Successivamente, i wrapper forniti direttamente da CityLearn rendono possibile all'agente di interfacciarsi con il nuovo ambiente discretizzato. Inoltre, l'implementazione prevede un decremento dell'epsilon ad ogni step. Questo permette, nelle prime fasi, di stimolare l'esplorazione dell'agente, per poi, man mano, ridurre l'esplorazione e sfruttare al massimo i Q-values ottenuti durante l'esplorazione. Il decremento dell'epsilon elimina la necessità di valutare la convergenza alla soluzione ottimale, poiché, arrivati al valore minimo di epsilon, la fase di esplorazione diventa trascurabile.
Nonostante ciò, ad ogni passo, attraverso la funzione *predict()* (funzione che restituisce l'azione per l'osservazione corrente), è possibile forzare l'*exploit* tramite il flag *deterministic*, obbligando l'agente a sfruttare la Q-table. Oltre a queste accortezze, l'algoritmo segue a grandi linee lo pseudocodice descritto in precedenza.
L'algoritmo viene parametrizzato come segue:

- **epsilon** (default: 1.0): Tasso di esplorazione.

- **minimum_epsilon** (default: 0.01): il valore minimo a cui il tasso di esplorazione può decrementare.
- **epsilon_decay** (default: 0.0001): tasso di decremento esponenziale di `epsilon`.
- **learning_rate** (default: 0.05): corrisponde all'α dell'equazione di Bellman, cioè il tasso di apprendimento.
- **discount_factor** (default: 0.90): corrisponde al π dell'equazione di Bellman, cioè il fattore di sconto.
- **q_init_value** (default: np.nan): valore di inizializzazione della Q-Table.

Valutazione La valutazione del Tabular Q-Learning risulta impossibile. Ciò è dovuto principalmente al problema della discretizzazione: sebbene le azioni siano in numero ridotto (massimo 5), risultando fattibile discretizzarle, lo stesso non vale per le osservazioni. Queste, infatti, sono molto numerose e, indipendentemente dal livello di discretizzazione scelto, risulta impossibile memorizzare tutte le combinazioni in una macchina normale.

Pertanto, si può affermare che le osservazioni in CityLearn non sono discretizzabili in modo efficiente, portandoci a considerare algoritmi che lavorano su spazi continui, oppure con spazi delle azioni discreti, come consente ad esempio il Deep Q-Learning.

8.3.2 Soft Actor-Critic

Il Soft Actor-Critic (SAC) è un algoritmo di Deep Reinforcement Learning caratterizzato dall'approccio off-policy e l'entropia. Inoltre, SAC è progettato per operare efficacemente con spazi d'azione continui, rendendolo ideale per ambienti complessi come CityLearn.

Concetti Teorici Il Soft Actor-Critic (SAC) si basa su una combinazione di principi tipici del Deep Reinforcement Learning, con alcune caratteristiche distintive tra cui l'entropia, che definisce la casualità della policy (più è elevata, più l'output della policy diventa imprevedibile). Infatti, mentre nel Reinforcement Learning standard l'obiettivo è di massimizzare la somma attesa dei reward, l'algoritmo si occupa inoltre di **massimizzare l'entropia**. Ciò permette di incoraggiare la policy a mantenere un alto grado di casualità, promuovendo l'esplorazione in maniera più robusta. Si ottiene così la nuova

formula per la policy ottimale:

$$\boldsymbol{\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho^\pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))]}$$

Dove:

- $\boldsymbol{\mathcal{H}(\pi(\cdot|s_t))}$ è l'entropia della politica, che misura quanto siano imprevedibili o casuali le azioni generate.
- $\boldsymbol{\alpha}$ è un parametro che definisce l'importanza dell'entropia rispetto alla ricompensa. Un $\boldsymbol{\alpha}$ più alto favorisce più esplorazione.

Il SAC utilizza inoltre un'**architettura Actor-Critic**, in cui una rete neurale (actor) impara una policy stocastica che genera azioni mantenendo un alto grado di casualità, mentre una o più reti neurali (critic) stimano il Q-Value valutando l'efficacia delle azioni eseguite. Questo approccio si ispira all'algoritmo di policy iteration, che alterna fasi di valutazione e miglioramento, in modo da consentire una continua evoluzione della policy. Il termine "soft" deriva dall'integrazione del concetto di massimizzazione dell'entropia nella valutazione delle azioni e del "soft" Q-value. La soft Q-function $\boldsymbol{Q_\theta(s_t, a_t)}$ viene definita combinando il reward immediato con il valore atteso del prossimo stato, adeguatamente penalizzato da un termine che dipende dalla log-probabilità dell'azione, limitato dal parametro $\boldsymbol{\alpha}$ (definito precedentemente) :

$$\boldsymbol{Q_\theta(s_t, a_t) = r(s_t, a_t) + \gamma\, \mathbb{E}_{s_{t+1}\sim p}[\, V_\theta(s_{t+1})]}$$

dove la funzione di valore $\boldsymbol{V_\theta(s_{t+1})}$ è definita come:

$$\boldsymbol{V_\theta(s_{t+1}) = \mathbb{E}_{a_{t+1}\sim\pi}[\, Q_\theta(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})]}$$

Questo approccio consente di bilanciare in modo efficace la ricerca di reward elevati, che attraverso l'utilizzo dell'entropia e di una adeguata esplorazione, evita di convergere ad una soluzione sub-ottimale.
Tuttavia, queste formule risultano funzionali esclusivamente per ambienti tabulari, portandoci all'implementazione del Soft Actor-Critic, che effettua una approssimazione della Q-function e della policy, attraverso l'utilizzo delle reti neurali e gradiente stocastico della architettura Actor-Critic, per permetterci di lavorare con spazi continui. Infatti, il SAC utilizza due diverse reti per le Q-Function, chiamate $\boldsymbol{Q_{\theta_1}}$ e $\boldsymbol{Q_{\theta_2}}$, ed una rete per la policy:

- **Due reti per la Q-function (Critic)**: L'uso di due reti serve a ridurre il rischio di sovrastima del valore della Q-function, che può portare a decisioni sub-ottimali. Ogni rete stima la Q-function in modo indipendente, e durante l'aggiornamento, si utilizza il minimo tra le due stime per evitare una sovrastima del valore delle azioni.

- **Rete per la policy (Actor)**: è una rete neurale che prende come input lo stato $\boldsymbol{s_t}$ e restituisce una distribuzione di probabilità sulle azioni, permettendo di non determinare sempre una azione fissa. La politica viene aggiornata per massimizzare l'entropia (promuovendo l'esplorazione) e il valore della ricompensa attesa.

Il gradiente stocastico permette di aggiornare i pesi delle funzioni con l'obiettivo di ridurre l'errore propagato durante l'addestramento. Viene definito stocastico perché, anziché calcolare il gradiente su tutti i dati, lavora su un sottoinsieme scelto casualmente. Infine, l'algoritmo adotta un approccio **off-policy**, impiegando un replay buffer per immagazzinare le esperienze raccolte durante l'interazione con l'ambiente, permettendo di aggiornare le reti anche utilizzando dati provenienti da politiche differenti da quella attuale.
Di seguito si trova l'algoritmo generale del SAC:

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

Implementazione Come per la discretizzazione nel Q-Learning, è utile la normalizzazione dei valori anche nell'algoritmo SAC, per gestire in modo più efficace la disparità delle scale, migliorando stabilità e prestazioni. Nella pratica, dopo una fase iniziale di addestramento utile a raccogliere statistiche, viene eseguita una normalizzazione periodica delle osservazioni, portandole ad una scala compresa tra -1 e 1. Tutto ciò viene implementato direttamente nell'agente, senza l'utilizzo di wrapper esterni, e consente di rendere i dati più adatti per l'input delle reti neurali. Inoltre, durante questa fase vengono rimosse le osservazioni non necessarie, come ad esempio la *solar_irradiance* nel caso in cui non siano presenti pannelli fotovoltaici, alleggerendo così il problema.
L'implementazione delle reti neurali e del replay buffer utilizzati nel modulo

citylearn.sac è descritta nel modulo *citylearn.rl*, che importa librerie PyTorch per facilitare lo sviluppo. Inoltre, il SAC è un implementazione del modulo *citylearn.rlc*, che definisce parametri generali di algoritmi di Reinforcement Learning e comportamenti comuni, come il tasso di apprendimento α o il discount. I valori principali di default in un RLC sono:

- **hidden_dimension (default: [256, 256])**: dimensione dei layer nascosti delle reti neurali.
- **discount (default: 0.99)**: fattore di sconto.
- **tau (default: 0.005)**: tasso di decadimento, utilizzato per l'aggiornamento delle reti.
- **alpha (default: 0.2)**: temperatura, termine di bilanciamento tra esplorazione ed esploitazione.
- **lr (default: 0.0003)**: tasso di apprendimento per l'ottimizzazione.
- **batch_size (default: 256)**: numero di campioni utilizzati per ogni layer.
- **replay_buffer_capacity (default: 100000)**: capacità del buffer di replay.
- **action_scaling_coefficient (0.5)**: coefficiente di scalatura delle azioni, utile alla normalizzazione.
- **reward_scaling (default: 5.0)**: coefficiente di scaling per i premi.
- **update_per_time_step (default: 2)**: numero di aggiornamenti per ogni passo temporale.

Per il resto, il codice segue strettamente la descrizione dello pseudocodice, aggiungendo la possibilità di forzare il determinismo con il flag *deterministic* nella funzione *predict* (in questo caso, scegliendo l'azione determinata dalla policy). Infine, viene implementata un versione del SAC che durante la fase di esplorazione, al posto di effettuare scelte casuali, esegue azioni determinate da un algoritmo RBC a piacimento. Se inizializzato l'agente SACRBC senza definire un RBC, effettuerà comunque scelte casuali, rendendolo identico al SAC base.

Valutazione A differenza degli algoritmi RBC, l'algoritmo Soft Actor-Critic può essere valutato da diversi punti di vista:

- **Reward**: Essendo un algoritmo di Reinforcement Learning, la scelta della funzione di reward incide significativamente sulle performance. In questo caso, si considerano principalmente le seguenti funzioni di reward: la *Base Reward Function*, che si interessa al consumo, la *Comfort Reward*, che pone l'accento sulla riduzione del discomfort e la *Solar Penalty Reward* che si pone l'obiettivo di migliorare l'utilizzo dei panelli solari. Si vedrà alla fine come la soluzione migliore sia la combinazione di Solar Penalty Reward e Comfort Reward.

- **Algoritmo di Apprendimento**: Durante la fase di esplorazione, le azioni possono essere scelte in maniera casuale, oppure utilizzando un algoritmo RBC. Tuttavia, in molti casi gli algoritmi RBC tendono ad avere fra di loro poche variazioni nell'ambito del SAC, motivo per cui si preferisce considerare solamente la scelta casuale e l'*OptimizedRBC*.

Inoltre, oltre a considerare le cost_function, è interessante analizzare come variano i reward tra i vari episodi, in modo da osservare e valutare il processo di apprendimento dell'algoritmo SAC. È importante sottolineare che CityLearn implementa la funzione *learn* che, dato un numero di episodi e impostato il parametro *deterministic_finish*, esegue l'algoritmo normalmente per tutti gli episodi (aggiornando ed esplorando) e, nell'ultimo episodio, applica il determinismo, selezionando le azioni esclusivamente in base alla policy (privando dell'esplorazione). Nella pratica, si sceglie di effettuare sette episodi, in quanto risultano sufficienti a raggiungere un punto di convergenza.

Andamento dei Reward in SAC

Episodio	Reward Min	Reward Max	Reward Somma	Reward Media
1	-116.954224	0.0	-16119.357422	-22.419134
2	-71.661255	0.0	-8744.085938	-12.161455
3	-76.473007	0.0	-4638.866699	-6.451831
4	-74.766548	0.0	-3592.313965	-4.996264
5	-76.393280	0.0	-3432.064453	-4.773386
6	-78.283836	0.0	-2978.939941	-4.143171
7	-78.597725	0.0	-1531.421143	-2.129932

Andamento dei Reward in SAC RBC

Episodio	Reward Min	Reward Max	Reward Somma	Reward Media
1	-294.368896	-0.002338	-80293.781250	-111.674240
2	-75.671593	0.000000	-8860.140625	-12.322865
3	-79.719688	0.000000	-5977.074219	-8.313038
4	-72.246971	0.000000	-5100.895020	-7.094430
5	-47.176891	0.000000	-3960.461914	-5.508292
6	-56.726280	0.000000	-3490.077148	-4.854071
7	-69.394058	0.000000	-1507.924194	-2.097252

In questo caso, la scelta di un reward rispetto a un altro influisce poco sull'andamento complessivo. Analizzando la media, si osserva un decremento costante fino all'episodio 7 (evidenziato in blu), dove avvengono scelte deterministiche. Questo comportamento suggerisce che l'algoritmo SAC tende a ridurre il reward medio nel corso degli episodi fino a raggiungere la convergenza, per poi nel finale applicare scelte deterministiche, in cui entrambe le varianti presentano una lieve variazione fra loro. Inoltre, si può notare come l'algoritmo SAC puro operi in modo più efficiente rispetto alla variante SAC con RBC. Infatti, l'esplorazione senza vincoli del SAC puro permette all'agente di sperimentare un numero maggiore di azioni, aumentando così la probabilità di identificare scelte migliori durante il processo di apprendimento. Al contrario, l'introduzione dell'RBC limita significativamente l'esplorazione, vincolando l'agente a seguire un percorso più rigido e riducendo la possibilità di scoprire soluzioni ottimali. Nonostante ciò, si parla comunque di una differenza abbastanza lieve, che porta comunque entrambi alla convergenza.
Per quanto riguarda il confronto con l'algoritmo di Baseline, entrambe le varianti di SAC ottengono risultati relativamente buoni rispetto al reward scelto, superando le prestazioni della baseline. Da sottolineare che i valori delle tabelle seguenti fanno riferimento esclusivamente all'ultimo episodio eseguito con determinismo.

Base Reward Function

SAC - Base Version

name cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.450	0.485	0.537	0.491
discomfort_proportion	0.947	0.869	0.923	0.913
electricity_consumption_total	0.455	0.493	0.543	0.497

SAC - RBC Version

name cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.441	0.518	0.571	0.510
discomfort_proportion	0.972	0.940	0.920	0.944
electricity_consumption_total	0.444	0.527	0.574	0.515

In questo caso, si può notare come la *Base Reward Function* abbia ridotto a metà i consumi rispetto all'algoritmo di baseline, in quanto penalizzata esclusivamente su quelli. Tuttavia, viene trascurato il discomfort, che risulta eccessivo poiché completamente ignorato dalla funzione di reward. Il comportamento per la gestione dei dispositivi di climatizzazione sono da ritenere totalmente casuali, in quanto non appresi attraverso dei reward ma attraverso l'esplorazione casuale e appresi di conseguenza. Questa assenza di vincoli e gestione, è caratteristica di tutte le varianti che non si preoccupano del discomfort. Aumentando il numero di episodi di apprendimento si può notare come si arrivi ad diminuire sempre di più l'utilizzo dei dispositivi di climatizzazione.

Comfort Reward Function

SAC - Base Version

name / cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.798	1.056	0.860	0.904
discomfort_proportion	0.142	0.062	0.012	0.072
electricity_consumption_total	0.799	1.053	0.864	0.905

SAC - RBC Version

name / cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.771	0.969	0.903	0.881
discomfort_proportion	0.095	0.078	0.007	0.060
electricity_consumption_total	0.780	0.973	0.907	0.887

In questo caso, abbiamo ridotto notevolmente il discomfort, ma a costo di un aumento dei consumi, che risultano comunque accettabili considerato il confronto con gli altri algoritmi. I consumi sono risultato di un utilizzo più moderato dei dispositivi di climatizzazione, a differenza degli algoritmi RBC che ne facevano un utilizzo eccessivo.

Solar Penalty Reward Function

SAC - Base Version

name / cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.534	0.592	0.579	0.568
discomfort_proportion	0.778	0.731	0.915	0.808
electricity_consumption_total	0.539	0.603	0.587	0.576

SAC - RBC Version

name / cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.517	0.650	0.722	0.630
discomfort_proportion	0.822	0.437	0.388	0.549
electricity_consumption_total	0.520	0.664	0.731	0.639

Come per il Base Reward Function, il Solar Penalty Reward Function ottiene dei consumi ridotti pagando con un alto livello di discomfort. Ciò è dovuto perché questo reward incita l'agente a ridurre i consumi e massimizzare utilizzo dell'energia solare, tralasciando però le temperature. Una differenza sostanziale con il Reward Function è un maggiore utilizzo generale delle batterie. Infatti, l'utilizzo delle batterie può essere visto come un consumo di elettricità che se non gestito correttamente con i giusti rilasci, può essere uno spreco. Con il Solar Penalty Reward, il corretto utilizzo incita l'agente a gestire le batterie, a differenza del Base Reward che non le utilizza affatto.

Solar Penalty and Comfort Reward Function

SAC - Base Version

name cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.784	0.994	0.884	0.887
discomfort_proportion	0.123	0.052	0.003	0.060
electricity_consumption_total	0.782	1.003	0.889	0.891

SAC - RBC Version

name cost_function	Building_1	Building_2	Building_3	District
carbon_emissions_total	0.719	0.916	0.847	0.827
discomfort_proportion	0.240	0.127	0.017	0.128
electricity_consumption_total	0.726	0.923	0.853	0.834

Questa tipologia di reward combina i due reward precedenti, restituendo un buon compromesso tra discomfort ridotto e consumo moderato, risultando per l'ambiente una soluzione intermedia. A livello pratico, presenta un risultato molto simile al Comfort Reward, con un leggero miglioramento dei consumi.
Nel complesso, considerando le varianti che gestiscono il discomfort, SAC risulta un algoritmo valido per risolvere questa tipologia di problema, mantenendo delle prestazioni migliori all'algoritmo di baseline. Risulta però ancora non ottimale, in quanto reward e algoritmi model-free, non sono sufficienti a fornire un comportamento adatto e coerente in tutte le situazioni, presentando molto spesso degli errori e delle inefficienze. Ciò viene evidenziato nell'analisi della CityLearn Challenge 2023.

8.4 Community-Based Hierarchical Energy Systems Coordination Algorithm

L'algoritmo **CHESCA** (Community-Based Hierarchical Energy Systems Coordination Algorithm) è stato la soluzione alla CityLearn Challenge 2023, che, per quanto riguarda la Control Track (sezione della sfida che richiedeva di sviluppare degli agenti di controllo), ha ottenuto i risultati migliori. Il suo obiettivo era di perfezionare l'utilizzo di dispositivi di raffreddamento, batterie e accumulatori delle acque sanitarie, proponendosi come soluzione ai problemi di overfitting degli altri concorrenti. Questo algoritmo è ottimizzato per migliorare i parametri considerati nella sfida, i quali sono le seguenti Cost Function:

- ramping_average
- daily_one_minus_load_factor_average
- daily_peak_average
- all_time_peak_average
- one_minus_thermal_resilience_proportion
- annual_normalized_unserved_energy_total
- discomfort_proportion
- carbon_emission_total

Implementazione Il punto chiave di questo algoritmo è operare su più livelli:

- **Building Level**: si utilizza un controllore per ogni edificio, con l'obiettivo principale, tramite l'uso di previsioni, di minimizzare i valori di tutte le cost function.
- **Community Level**: lo scopo principale di questo livello è ottimizzare le cosiddette **grid cost function**, che rappresentano il comportamento complessivo del distretto, agendo anche sui singoli edifici. Le grid cost function sono legate al consumo della rete elettrica e alle sue variazioni nel tempo.

Questi due livelli comunicano tra loro e non sono indipendenti. Infatti, il community level fornisce al building level le azioni calcolate in base al grid cost, con l'obiettivo di dare direttive sui consumi. Successivamente, queste azioni vengono adattate alle esigenze dei singoli edifici, che a loro volta restituiscono le proprie azioni. In questo modo si bilanciano le necessità individuali degli edifici con quelle dell'intero distretto, ottimizzando sia su parametri locali che complessivi.

Inoltre, il building level è articolato in più moduli:

- **Temperature Controller**: regola il discomfort a ogni step mediante un controllore automatico **PID** (Proportional-Integral-Derivative). Il PID reagisce proporzionalmente all'errore, valuta l'accumulo dell'errore nel tempo e considera la velocità di variazione dell'errore, correggendo di conseguenza l'utilizzo dei dispositivi di raffreddamento.

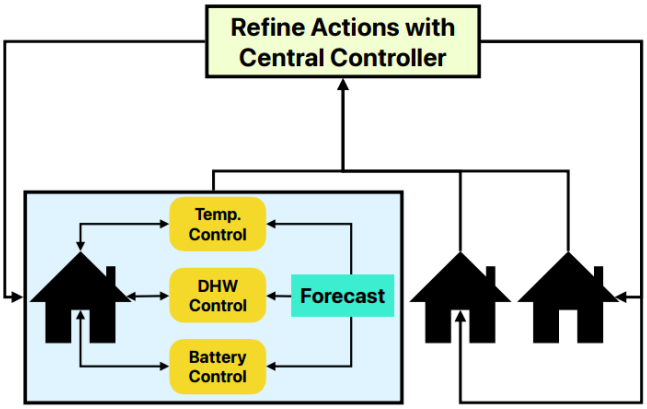
- **DHW Controller**: regola, tramite un controllo **RBC Hour-Based**, l'uso dell'accumulatore per l'acqua calda sanitaria. In sintesi, accumula energia durante le ore serali (quando la rete è meno sovraccarica) o quando è prevista una produzione di energia.

- **Electricity Storage Controller**: basandosi su previsioni della generazione solare e dei consumi, utilizza un algoritmo **A*** per pianificare una sequenza di azioni di carica/scarica della batteria discretizzate con una granulità g. Inoltre, la sequenza di azioni che minimizza il costo, si assicura che lo stato di carica della batteria rimanga sempre tra i limiti inferiori e superiori, dipendenti dall'ora del giorno.

Tutti questi moduli dipendono da previsioni a breve e lungo termine, che influenzano i loro comportamenti. Le previsioni vengono effettuate su temperatura, generazione solare, domanda di acqua calda e domanda dei dispositivi essenziali, permettendo anche di stimare i consumi di rete. Tutto ciò è reso possibile dall'utilizzo di **XGBoost**, una libreria di machine learning supervisionato che implementa il gradient boosting, che fornisce un ottimo modello di regressione.
È da precisare che i comportamenti cambiano in caso di interruzione di corrente, avviando una modalità denominata **Outage Mode**. In tale eventualità, l'algoritmo disattiva tutti gli accumulatori DHW, lasciando spazio a quelli essenziali, corregge il controllore delle temperature e ottimizza l'utilizzo delle batterie in base alla situazione.
Per il community level è presente un singolo modulo:

- **Central Controller**: si occupa della gestione del **grid cost**, con due obiettivi principali:

– **Consumption Smoothing**: si allinea il consumo energetico totale della comunità a una traiettoria media, con una certa banda di tolleranza. Se si prevede un consumo sopra al limite, si riduce prima il riscaldamento dell'acqua sanitaria, poi la capacità di raffreddamento fino a un limite massimo dettato da ogni singolo edificio. Se invece il consumo scende sotto il limite, si aumenta il riscaldamento dell'acqua e si caricano le batterie.
– **Peak Demand Reduction**: ridurre i picchi di domanda. Un picco si identifica quando il consumo supera di due deviazioni standard la media. In questo caso si riduce o si interrompe il raffreddamento fino a riportare il consumo entro i limiti.



Di seguito si trova lo pseudocodice che sintetizza il comportamento dell'algoritmo:

Algorithm 1 Pseudo-code of CHESCA.

for each time step $t \in \{1, \ldots, T\}$ **do**
  input: Observation vector $\boldsymbol{o}_t$, Set of action initializations $\boldsymbol{a}_t$
  while Stopping Criteria Not Met **do**
    predict: Update forecast variables $\widehat{\boldsymbol{y}}_t$
    for each building b in the set of buildings $\mathcal{B}$ **do**
      $a_t^{\theta}[b] \leftarrow \textsc{ControlTemperature}(\boldsymbol{o}_t, \widehat{\boldsymbol{y}}_t, \boldsymbol{a}_t)$
      $a_t^{w}[b] \leftarrow \textsc{ControlDHW}(\boldsymbol{o}_t, \widehat{\boldsymbol{y}}_t, \boldsymbol{a}_t)$
      $a_t^{e}[b] \leftarrow \textsc{OptimizeBatteryUsage}(\boldsymbol{o}_t, \widehat{\boldsymbol{y}}_t, \boldsymbol{a}_t)$
    end for
    $\boldsymbol{a}_t \leftarrow \textsc{Aggregate}(a_t^{\theta}[b], a_t^{w}[b], a_t^{e}[b] \; \forall \; b \in \mathcal{B})$
    $\boldsymbol{a}_t \leftarrow \textsc{CentralController}(\boldsymbol{o}_t, \widehat{\boldsymbol{y}}_t, \boldsymbol{a}_t)$
  end while
  output: $\boldsymbol{a}_t$
end for

8.5 Multi-Agent Reinforcement Learning using Independent SAC (MARLISA)

Un algoritmo che consente un apprendimento multi-agente coordinato, utilizzando più agenti RLC che operano in modo decentralizzato ma condividono informazioni per un obiettivo comune.

Riferimenti

- Documentazione CityLearn: https://www.citylearn.net/
- Tutorial di utilizzo di CityLearn: https://colab.research.google.com/github/climatechange-ai-tutorials/citylearn/blob/main/citylearn_ccai_tutorial.ipynb
- Documentazione sull'algoritmo SAC: https://arxiv.org/pdf/1812.05905
- Documentazione sull'algoritmo CHESCA: https://hal.science/hal-04685791v1/document