Maze Solving Algorithms through Stackelberg Model-Based Reinforcement Learning

Algoritmi di Risoluzione di Labirinti attraverso Model-Based Reinforcement Learning nella formulazione di Stackelberg

---

Giulia Ghisolfi, g.ghisolfi@studenti.unipi.it, 664222

Master Degree in Computer in Science (Artificial Intelligence Curriculum), University of Pisa
Algorithmic Game Theory course (211AA),
Academic Year: 2022-2023

Outline

- Introduzione
  - Obiettivo del progetto
  - Conoscenze preliminari
  - Lavori correlati
- Formulazione del problema
  - Giocatori
  - Algoritmi
  - Equilibri di Stackelberg
- Implementazione ed esperimenti
  - Implementazione
  - Esperimenti e Risultati
- Conclusioni

Introduzione

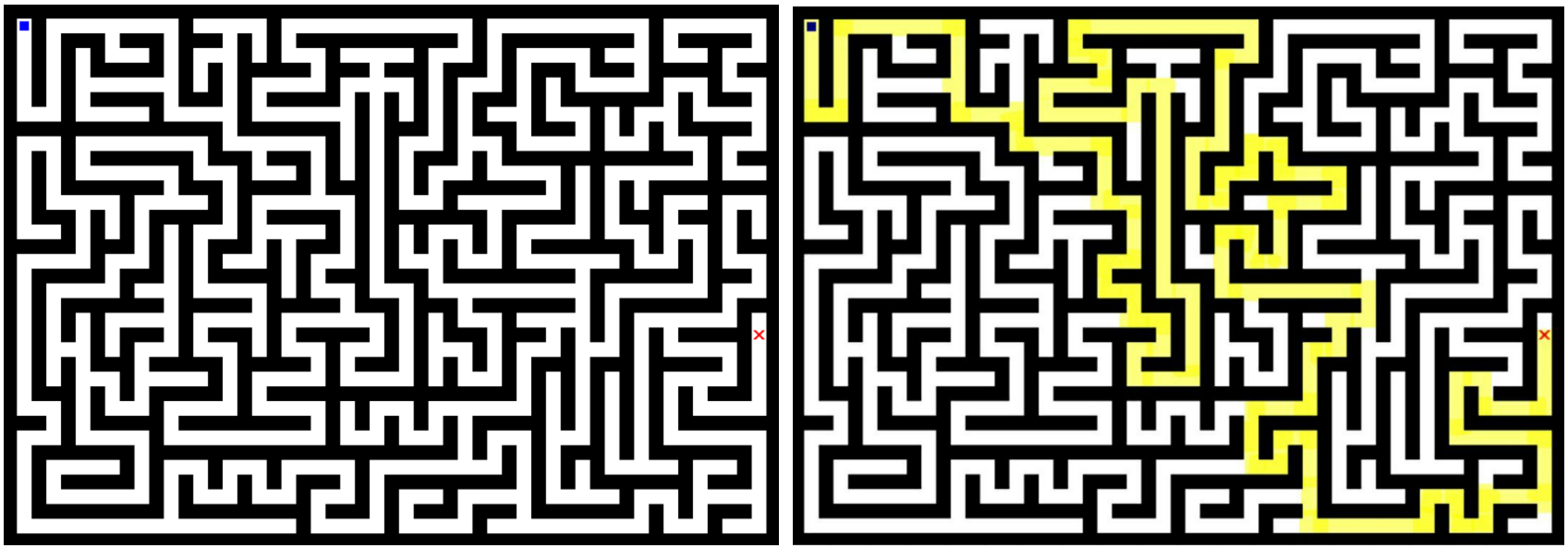
Obiettivo

Maze solving: metodi automatizzati per la risoluzione dei labirinti.

L'obiettivo è generare un agente in grado di apprendere regole generali che consentano **risoluzione di labirinti** precedentemente non visti e di cui non ha conoscenza a priori **in modo efficiente**.

Obiettivo

L'obiettivo è generare un agente in grado di apprendere apprendere regole generali che consentano **risoluzione di labirinti** precedentemente non visti e di cui non ha conoscenza a priori **in modo efficiente**.



Obiettivo

Questo lavoro esplora l'applicazione di algoritmi Model-Based di Reinforcement Learning alla risoluzione di labirinti.

L'agente è formulato nella sua forma di Stackelberg, rappresentando un gioco asimmetrico a due giocatori.

Obiettivo

L'agente è formulato come due giocatori distinti che operano in una dinamica avversaria, cercando di determinare l'equilibrio del duopolio di Stackelberg.

Conoscenze preliminari: Fonti

Steven Tadelis
Game theory: An Introduction
Princeton university press [4]

Richard S. Sutton and Andrew G. Barto
Reinforcement Learning: An Introduction
The MIT Press [3]

Duopolio di Stackelberg

Gioco ad informazione perfetta che rappresenta una variazione a mosse sequenziali del duopolio di Cournot.

Illustra che l'ordine delle mosse è importante, e gli agenti razionali dovranno tenere conto di questo aspetto.

Duopolio di Stackelberg

Due agenti sono coinvolti: il leader e il follower.

Il leader ha un vantaggio strategico in quanto decide la prima mossa e determinare la propria strategia prima che il follower prenda la sua decisione.

Lo scopo del leader è massimizzare i suoi guadagni, tenendo conto della risposta razionale che ci si aspetta dal follower.

Il follower, d'altra parte, risponderà in modo ottimale alle azioni del leader.

Duopolio di Stackelberg

Definition (Equilibrio di Stackelberg)

Sia $x_1^* \in S_1$, l'insieme di strategie del primo giocatore, una soluzione di Stackelberg se

$$x_1^* \in argmax\{u_1(x_1, R_2(x_1)) : x_1 \in S_1\}$$

$(x_1^*, R_2(x_1^*))$ è un equilibrio di Stackelberg se x_1^* è una soluzione di Stackelberg.

Dove $R_2(x_1^*) \in S_2$ rappresenta la miglior risposta del follower alla strategia del leader.

Reinforce Learning

Reinforce Learning è un paradigma del machine learning in cui un agente impara a prendere decisioni sequenziali interagendo con un ambiente.

L'agente compie azioni per massimizzare un segnale di ricompensa cumulativo nel tempo.

L'ambiente viene formalizzato attraverso il concetto di Markov Decision Process che rappresenta una formulazione diretta del problema.

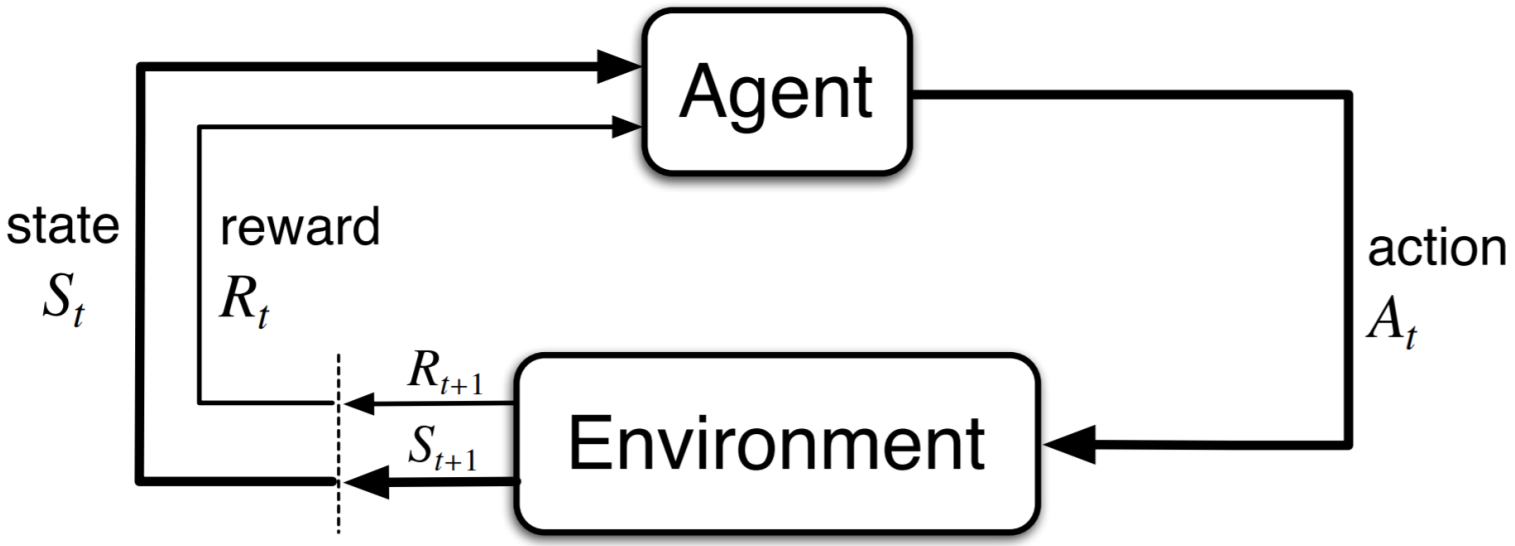


Figure 1: Interazione agente-ambiente in un MDP

Definition (Markov Decision Process)
Un Markov Decision Process è una tupla $M = \langle S, A, \mathbf{P}, R, \gamma \rangle$

S: spazio degli stati

A: spazio della azioni

$\mathbf{P}$: matrice di transizione tra stati, t.c.
$P^a_{ss'} = P(S_{t+1} = s' | S_t = s, A_t = a)$

R: funzione dei reward, t.c.
$R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$

γ: coeficiente di sconto, $\gamma \in [0, 1]$

Una policy è una mappatura dallo spazio degli stati a una distribuzione di probabilità sullo spazio della azioni:

$$\pi : S \to P(A)$$

Policy Player

Agente responsabile di prendere decisioni ed interagire con l'ambiente in base alla policy appresa.

Il suo scopo è quello di massimizzare i reward cumulativi.

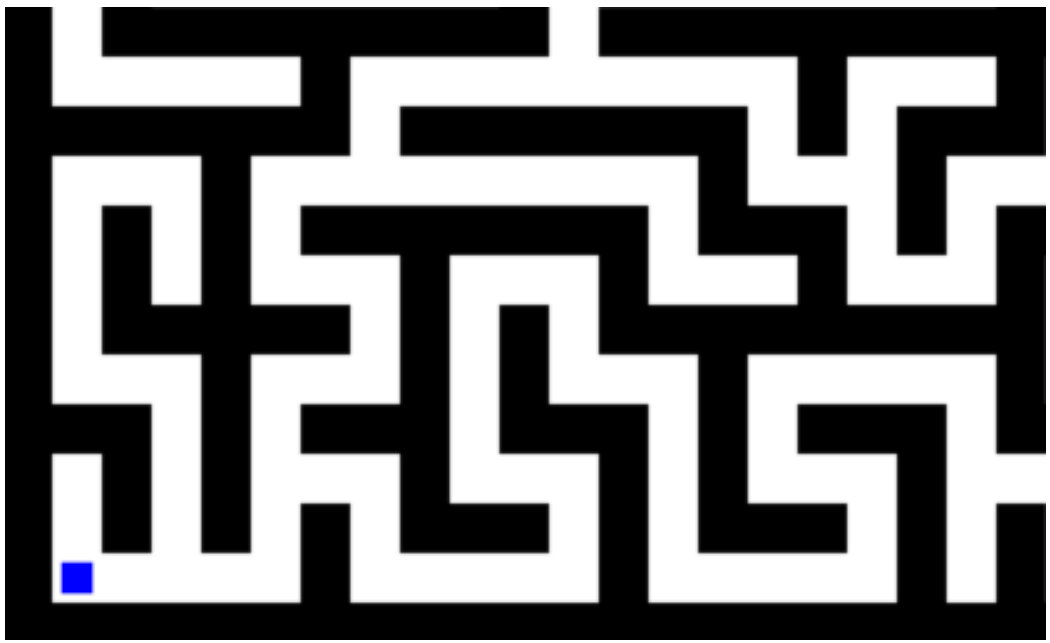
Definition (Rendimento Atteso Scontato)
Il rendimento atteso scontato è la somma cumulativa delle ricompense scontate del fattore γ al tempo t

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Model Player

Agente che simula una modello dell'ambiente creando una rappresentazione di esso in base ai segnali che riceve dall'ambiente stesso.

La rappresentazione dell'ambiente così generata è utilizzata per pianificare le future azioni.



Lavori correlati

Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar
A game theoretic framework for model based reinforcement learning
International Conference on Machine Learning (ICML) 2020 [2].

Dove vengono presentati algoritmi stabili ed efficienti model-based reinforcement learning.

Lavori correlati

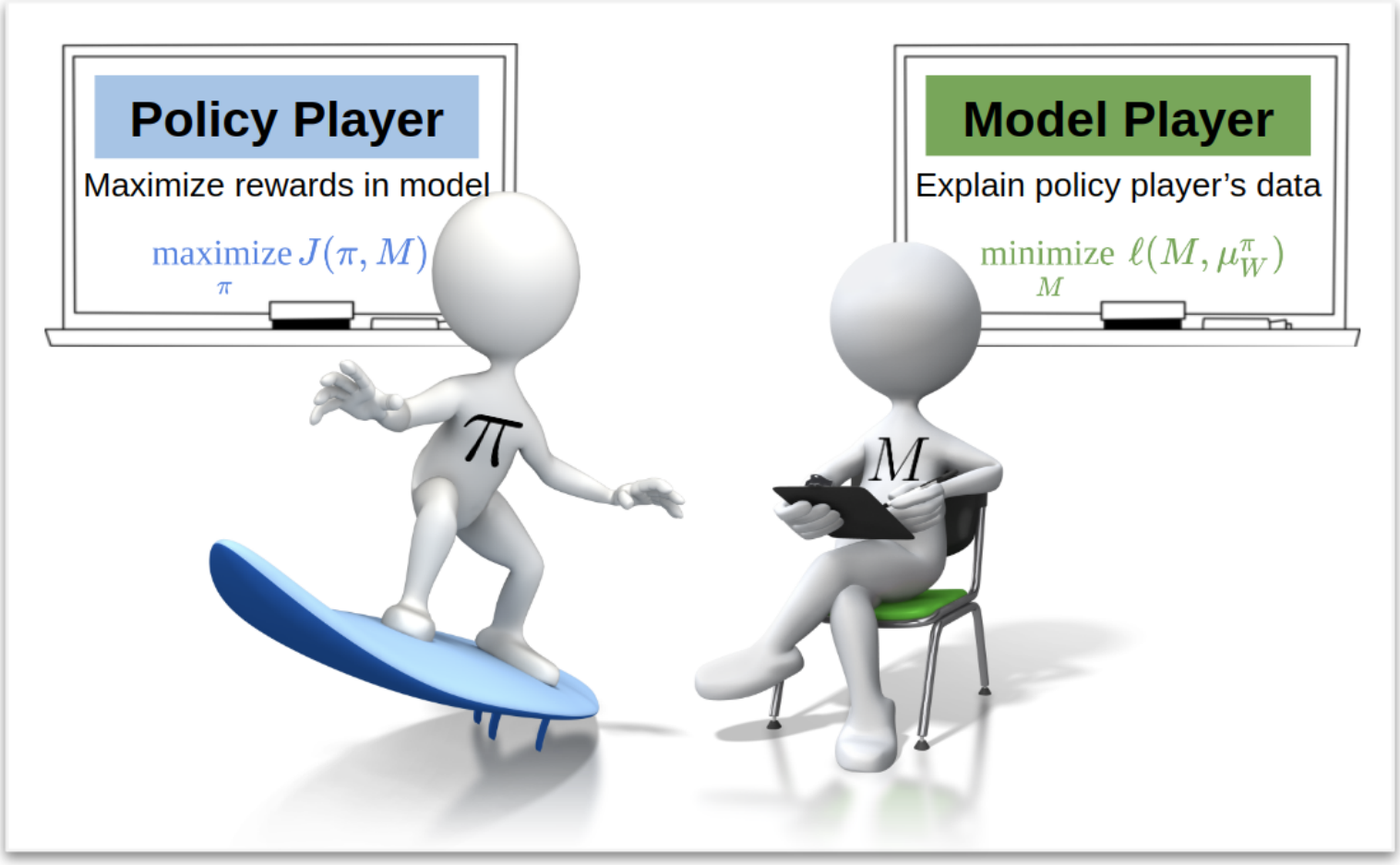


Figure 2: Policy and Model Palyers for Stackelberg formulation for Model-Based Reinforcement Learning

Lavori correlati

Negli algoritmi presentati il leader sceglie un approcio conservativo per aggiornare la propria strategia. Il follower, invece, adotta una strategia aggressiva.

Si sviluppano quindi due possibili scenari:

- Policy As Leader (algoritmo PAL)
- Model As Leader (algoritmo MAL)

Lavori correlati

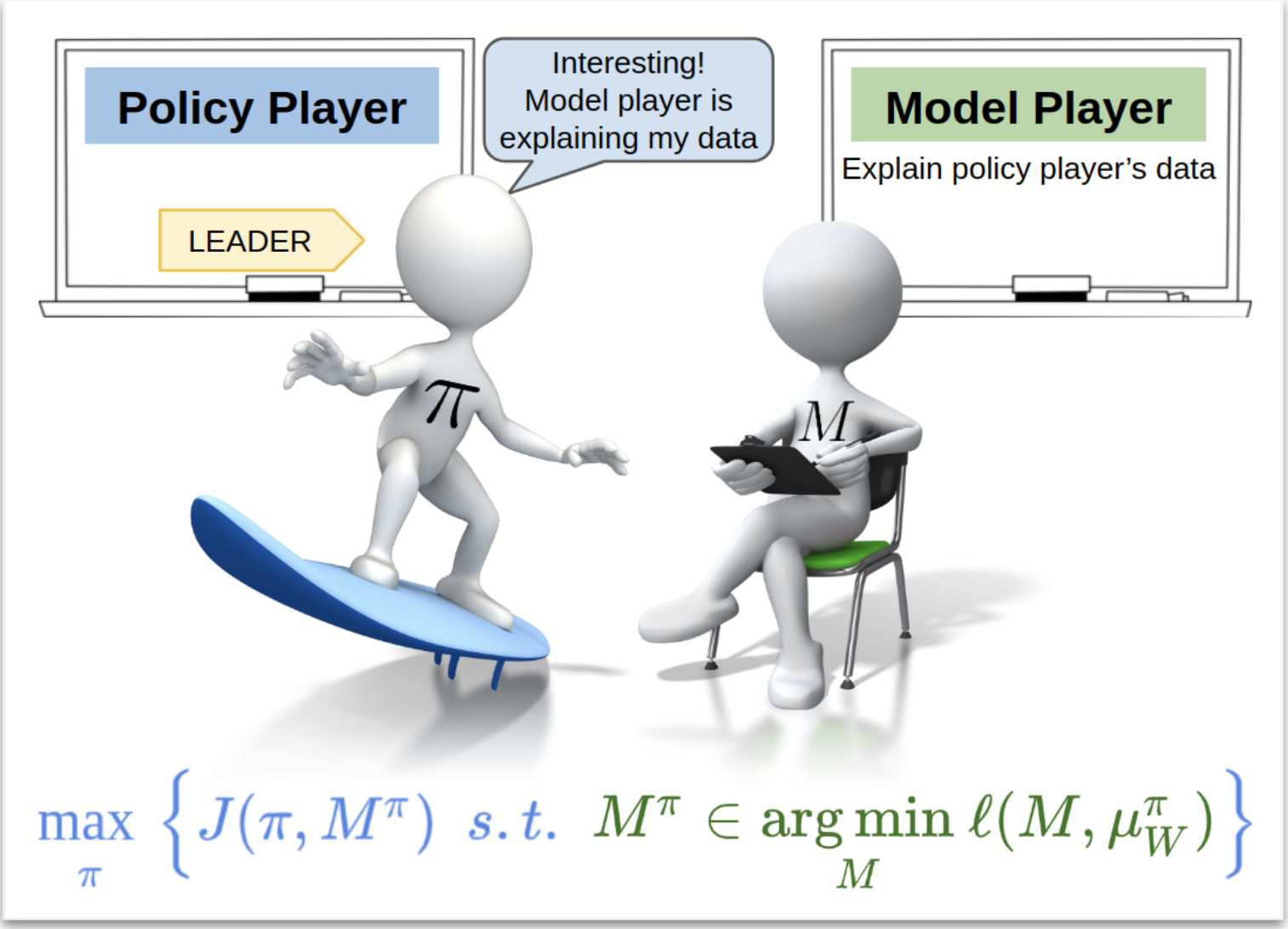


Figure 3: Policy As Leader.

Lavori correlati

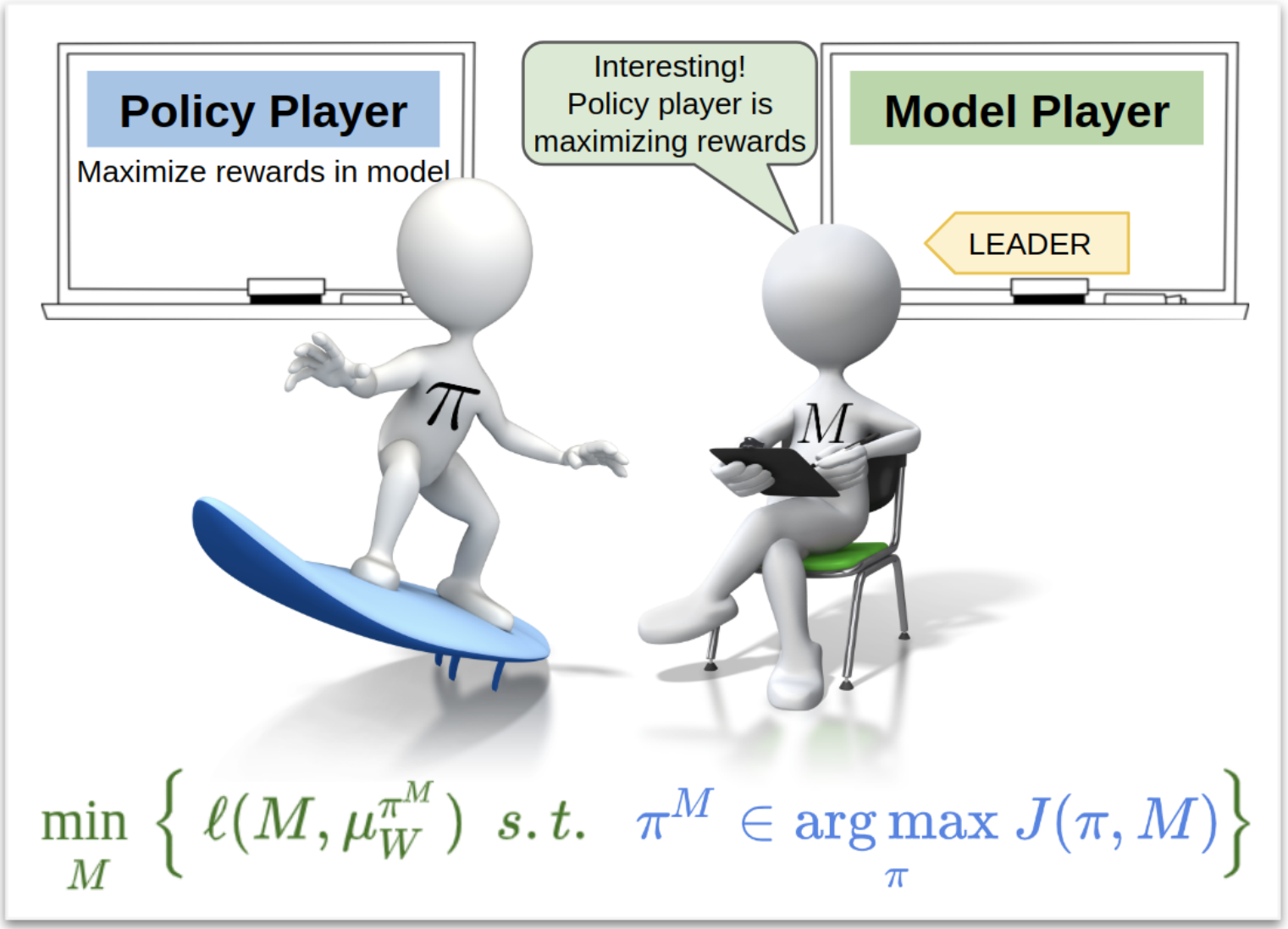


Figure 4: Model As Leader.

Formulazione del problema

Obiettivo

Derivare una policy generale che renda l'agente in grado di esplorare efficacemente labirinti non conosciuti.

Agente

L'agente che interagisce con l'ambiente viene modelizzato come due giocatori distinti che interagiscono in una dinamica avversaria.

I due giocatori osservano l'ambiente da due punti di vista distinti, apprendendo informazioni diverse dall'interazione con esso.

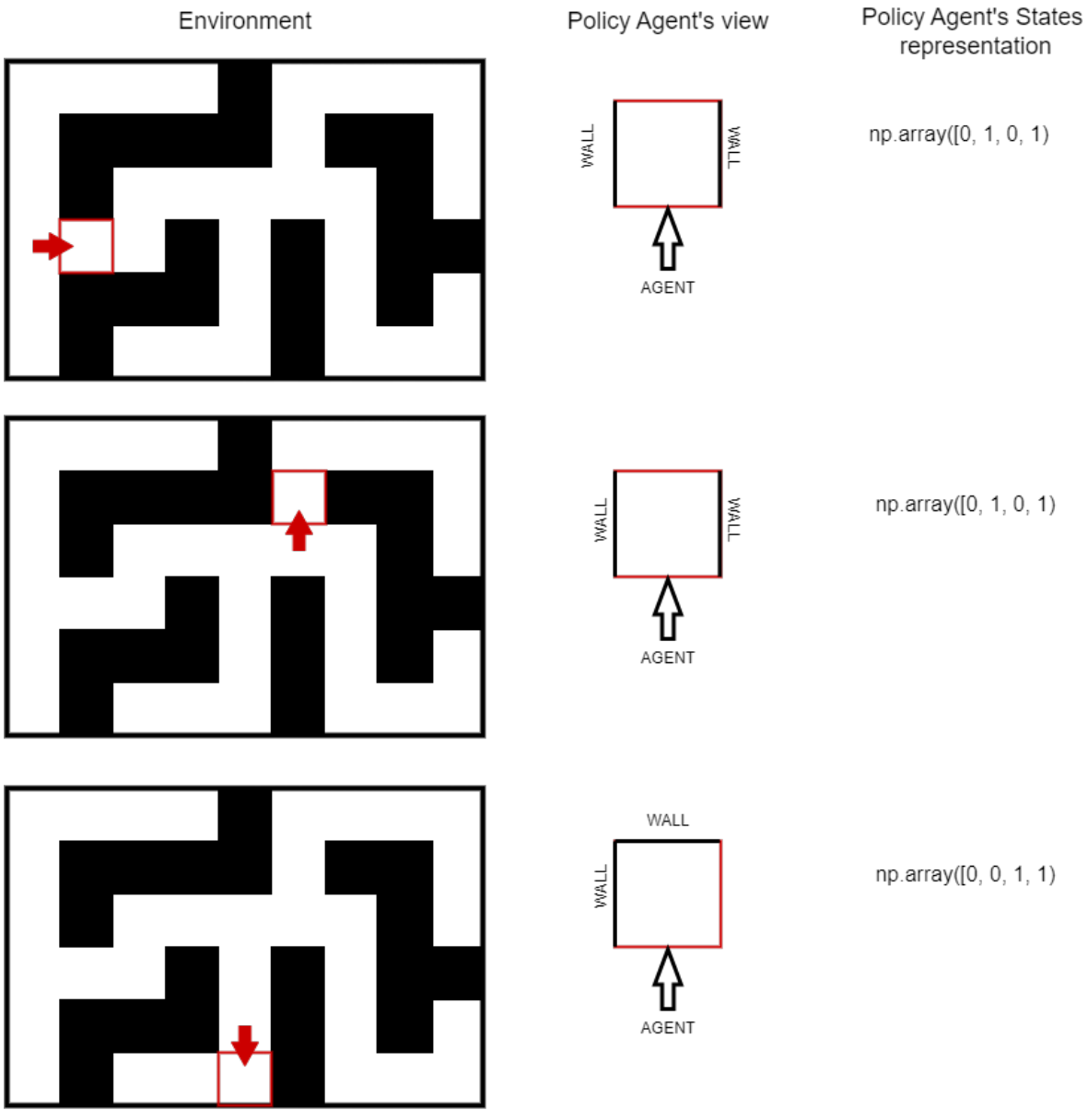
Agente: Policy Player

Rappresenta la policy, dal punto di vista di un osservatore relativo.

Lo spazio degli stati è formulato indipendentemente dalla posizione assoluta.

La policy così appresa è generalizzabile ed applicabile a diversi ambienti.

Agente: Policy Player



Agente: Model Player

Rappresenta i parametri del MDP.

Crea una rappresentazione specifica dell'ambiente che sta esplorando, la conoscenza appresa non è quindi generalizzabile.

Il suo scopo è quello di rendere interpretabili per un ambiente specifico i dati del policy player.

Algoritmi

Il problema è formulato come un duopolio di Stackelberg per facilitare l'apprendimento cooperativo tra policy e model player.

Abbiamo un problema di ottimizzazione bi livello, in cui ogni giocatore ha la sua funzione obiettivo:

Policy Player:

$$\max_{\pi} \; J(\pi, M^{\pi})$$

Model Player:

$$\min_{M} \; loss(M, \pi)$$

Algoritmi

Policy Player:

$$\max_{\pi} J(\pi, M^{\pi})$$

i.e. trovare la policy π s.t. $\max \mathbb{E}[G_t | S_t = s]$

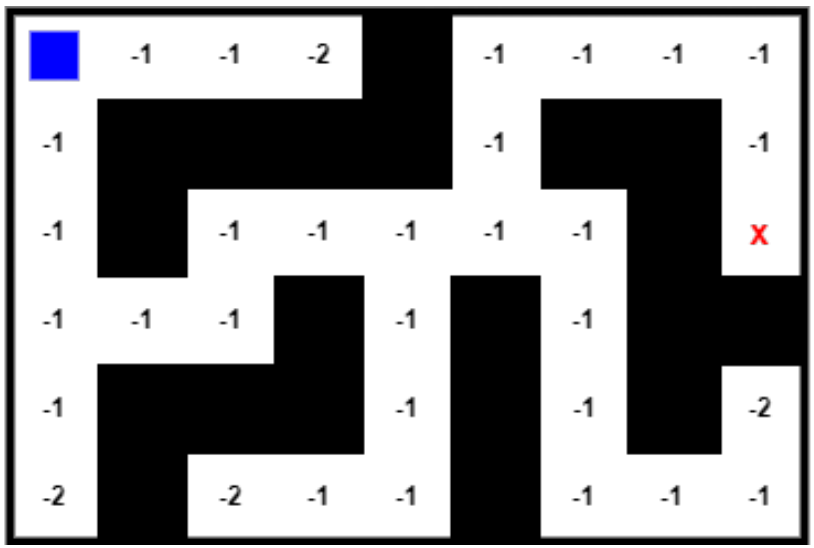


Figure 5: Reward ricevuti dall'ambiente per ogni stato del gioco

Definition (Value function)
La value functio di uno stato s seguendo la policy π, indicata con $v_\pi(s)$, è il rendimento atteso quando partendo da s e si seguendo la policy π fino alle stato terminale

$$v(s,\pi) = v_\pi(s) := \mathbb{E}[G_t|S_t = s,\pi] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s,\pi\right]$$

$$\forall s \in S, \text{ seguendo la policy } \pi$$

Una relazione d'ordine può essere definita sull'insieme delle policy.

$$\pi \geq \pi_0 \iff v_\pi(s) \geq v_{\pi_0}(s) \quad \forall s \in S$$

Ed esiste sempre almeno una policy che soddisfa questa condizione.

Definition (Value function ottimale)

$$v^*(s) := \max_\pi v_\pi(s) \, \forall s \in S$$

Definition (Action-value function)
La action-value function $Q(s, a)$ è il rendimento atteso per eseguire l'azione a dallo stato s seguendo la policy π

$$q(s,a) := \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a, \pi\right]$$
$$= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a, \pi\right]$$
$$\forall s \in S, \forall a \in A, \text{ seguendo la policy } \pi$$

$Q(s, a)$ ci consente di quantificare **l'efficacia di un'azione** a **in uno stato** s al tempo t, assumendo che dal tempo $t + 1$ in avanti, l'agente seguirà la policy π.

Definition (Action-value function ottimale)

$$q^*(s,a) := \max_{\pi} q_\pi(s,a) \quad \forall s \in S, \quad \forall a \in A.$$

Riscriviamo $q^*(s,a)$ in funzione di $v^*(s)$:

$$q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) \mid S_t = s, A_t = a]$$

Possiamo concludere che per trovare una policy che ottimizza il nostro obiettivo, dobbiamo cercare una **policy** corrispondente alla **action-value function ottimale**.

Algoritmi

Model Player:

$$\min_{M} \; loss(M, \pi)$$

Vogliamo **minimizzare** la **discrepanza tra due approssimazioni sequenziali dell'ambiente** da parte del modello M, a questo fine possiamo utilizzare $Q(s, a)$ come parametro del modello aggiornato attraverso gli episodi eseguiti sotto la policy nell'ambiente.

Utilizziamo l'Errore Quadratico Medio (MSE) come funzione di perdita, quantificando la discrepanza tra due approssimazioni sequenziali di Q ai passaggi temporali k e $k+1$.

Definition (Errore Quadratico Medio)

$$\text{MSE}(\hat{Q}^k, \hat{Q}^{k+1}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{Q}^k(s_i, a_i) - \hat{Q}^{k+1}(s_i, a_i))^2$$

dove N rappresenta il numero totale di coppie stato-azione nel dataset.

Algoritmi

Il dinamica tra i due giocatori è quindi modellizzata come un problema di massimizzazione e uno di minimizzazione

Questa formulazione permette una **dinamica cooperativa tra il modello e la policy**, dove gli **obiettivi** del follower e del leader **non sono in conflitto**, facilitando la convergenza degli algoritmi.

La decisione di formulare il problema in modo che il modello abbia come obiettivo la minimizzazione della loss evita che il modello generi rappresentazioni dell'ambiente irrisolvibili, arrestando il processo di apprendimento [1].

Algoritmi

La formulazione asimmetrica del gioco conduce quindi a due possibili versioni dell'algoritmo.

Policy As Leader:

Ottimizzazione del modello seguendo un approccio aggressivo

Aggiornamento della policy con startegia conservativa

Model As Leader:

Ottimizzazione della policy in modo aggressivo

Aggiornamento del modello con startegia conservativa

Algoritmi

Algorithm 1 Train Loop

1: **for** $k = 0, 1, 2, \ldots,$ max environments: **do**
2: **while** Stackelberg Equilibrium not found: **do**
3: Executing policy π in environment and collect episodes in D
Each episode consists of steps: (state, action, reward, next state)
4: update agent's parameters using MAL or PAL
5: **end while**
6: **end for**=0

Algoritmi: PAL

Ottimizzazione del modello: Q-learning [5]

$$Q(St, At) \leftarrow Q(St, At) + \alpha \left(R_{t+1} + \max_{a} Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

L'aggiornamento avviene attraverso i dati raccolti attraverso l'interazione della policy con l'ambiente.

La action-value function appresa approssima direttamente q^*, indipendentemente dalle policy seguita.

La convergenza di Q a q^* per un numero di step $n \rightarrow \infty$ è stata dimostrata dagli stessi autori [5].

Algoritmi: PAL

Aggiornamento della policy: Monte Carlo Policy Gradient

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

con θ parametrizzazione della policy e α come learning rate.

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s) q_\pi(s, a),$$

dove la distribuzione μ è la distribuzione on-policy sotto π.

Algoritmi: PAL

Inizializiamo un algoritmo stocastico di ascesa del gradiente:

$$\theta_{t+1} = \theta_t + \alpha \sum_a \hat{q}_\pi(S_t, a) \nabla \pi(a|S_t, \theta) = \theta_t + \alpha \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)}$$

Dai parametri θ aggiornati, otteniamo la policy π utilizzando una funzione softmax:

$$\pi(a|s, \theta) = \frac{e^{\theta^T \phi(s,a)}}{\sum_{a'} e^{\theta^T \phi(s,a')}}$$

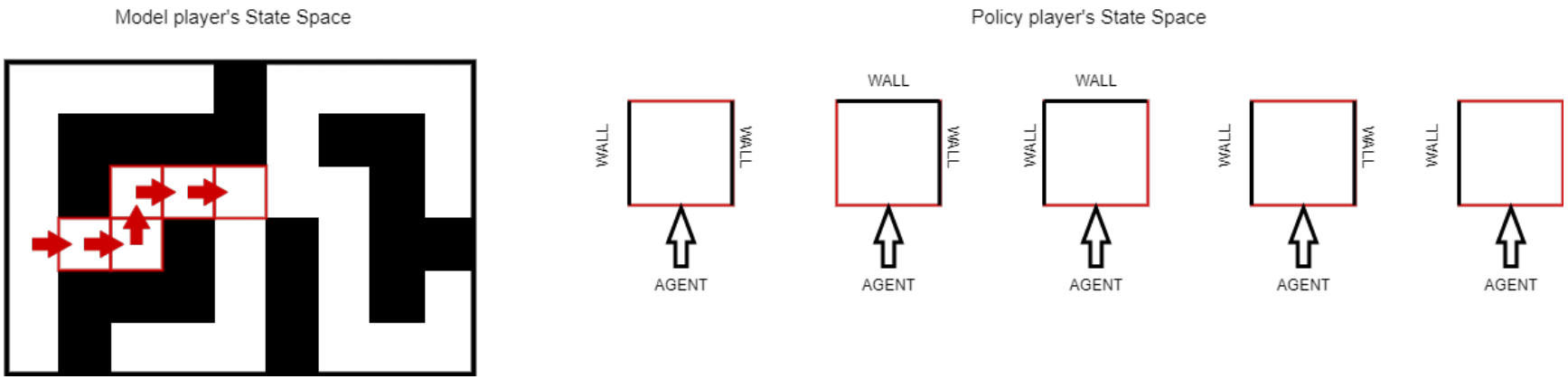
Dove $\phi(s, a)$ rappresenta la funzione delle caratteristiche stato-azione.

Algoritmi: MAL

Ottimizzazione della policy: Q-learning [5] (di nuovo!)

$$Q(St, At) \leftarrow Q(St, At) + \alpha \left(R_{t+1} + \max_{a} Q(S_{t+1}, a) - Q(S_t, A_t) \right)$$

L'aggiornamento avviene dall'interazione tra la policy e la rappresentazione dell'ambiente creata dal model player, mappando dallo spazio degli stati del model player a quello del policy player.



Algoritmi: MAL

La policy π viene derivata da Q utilizzando una funzione softmax:

$$\pi(a|s) = \frac{e^{q(s,a)}}{\sum_{a'} e^{q(s,a')}}$$

Questo approccio favorisce l'apprendimento bilanciando il mantenimento della conoscenza pregressa e l'aggiornamento dellla policy in base ai nuovi stati esplorati.

Aggiornamento del modello:
I parametri del modello vengono aggiornati utilizzando i dati raccolti durante l'esecuzione della policy nell'ambiente.

$$P(s,a) = \frac{\text{Numero di volte in cui } a \text{ è stata scelta da } s}{\text{Totale delle volte in cui } s \text{ è stato visitato durante l'episodio}}$$

e

$$\begin{aligned} q(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a, \pi\right] \\ &= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \cdot \max_{a'} q(s',a') \end{aligned}$$

Algoritmi: Equilibri di Stackelberg

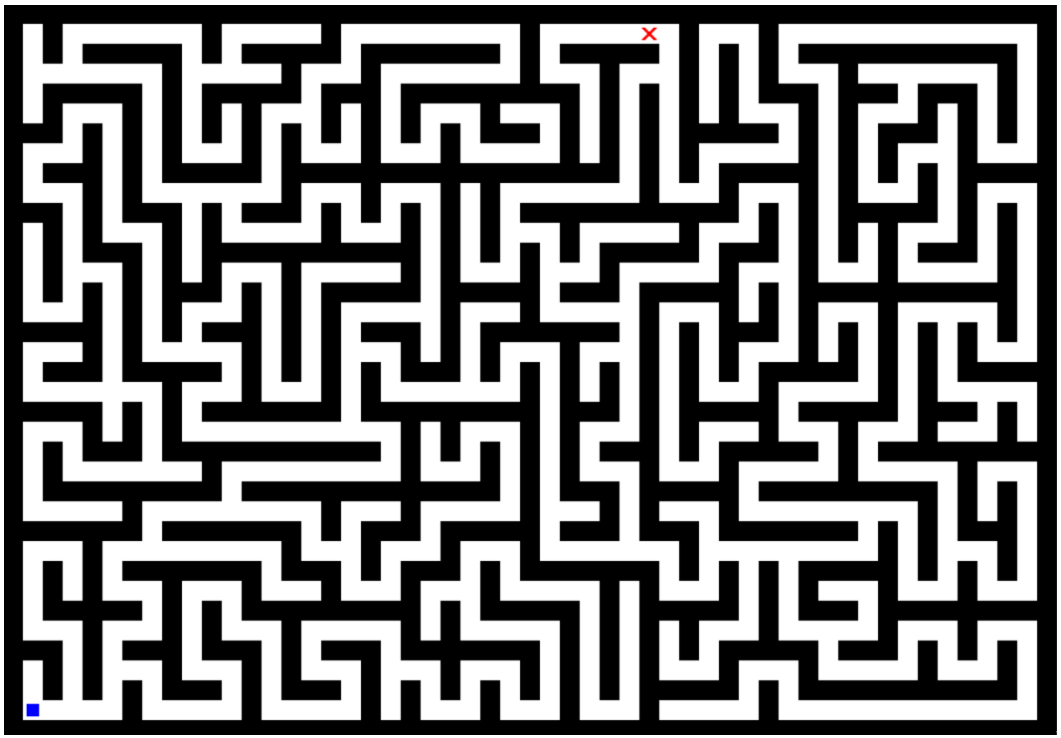
Alla conclusione di ogni interazione, il nostro obiettivo è determinare se le strategie scelte indipendentemente da ciascun giocatore, con l'intento di minimizzare o massimizzare i rispettivi obiettivi, siano in un equilibrio di Stackelberg.

Implementazione ed esperimenti

Environment

I labirinti sono generati randomicamente utilizzando un algoritmo che garantisce la creazione di labirinti aciclici e completamente connessi [6].

Definition (Spanning Tree)
Un grafo connesso non orientato che utilizza tutti i nodi e nel quale non ci sono circuiti.



Environment

Per l'implementazione dell'ambiente è stata usata la libreari **Matrix MDP gym** *Third-Party Environment* del Gymnasium, un'estensione della libreria Gym di OpenAI.

Agents

ModelAgent class: simula l'ambiente

Spazio della azioni A: discreto e finito, conosciuto a priori

Spazio degli stati S: discreto e finito, inizialmente non conosciuto, viene inizializzato con lo stato iniziale

contiene metodi per il rendering tramite PyGame

Agents

PolicyAgent class:

Spazio della azioni A: discreto e finito, conosciuto a priori

Spazio degli stati S: discreto e finito, inizialmente non conosciuto, viene aggiornato tramite l'esperienza

Policy π: inizializata con ϵ-greedy

Definition (ϵ-greedy policy for PolicyAgent)

$$\pi(a|s) = \begin{cases} \frac{\epsilon}{|A|} + 1 - \epsilon & \text{if } a \text{ is the first possible} \\ & \text{action agent's left} \\ \frac{\epsilon}{|A|} & \text{otherwise} \end{cases}$$

Esperimenti

Configurazioni di parametri utilizzate negli esperimenti, essi influenzano il comportamento e le prestazioni dell'algoritmo nelle attività di risoluzione di labirinti.

Parameter	**Value**
Maze Width	101
Maze Height	51
Discount Factor (γ)	0.995 (suggested in [2])
Learning Rate Q-learning	0.01
Learning Rate Gradient Ascent	Between 0.2 and 0.01
Number of Environments	Between 5 and 30
Maximum Iterations per Environment	10
Episodes per Iteration	3
Maximum Epochs per Episode	6000
Epsilon for Epsilon-Greedy	0.1

Table 1: Configurazioni di parametri utilizzate negli esperimenti

Risultati

MAL: convergere ad una distribuzione uniforme sullo spazio delle azioni.

PAL: risultati più promettenti, necessita di learning rate più bassi

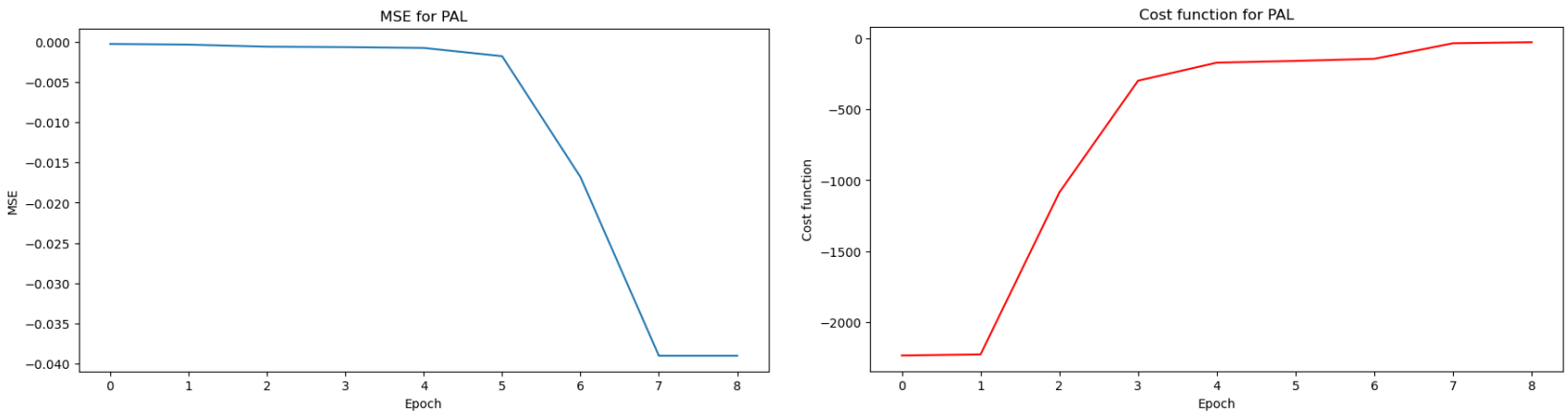


Figure 6: Learning curve per algoritmo PAL algorithm: Mean Squared Error (MSE) e policy Cost Function. Parametri: 5 environments, learning rate α pari a 0.01 per Monte Carlo.

Risultati

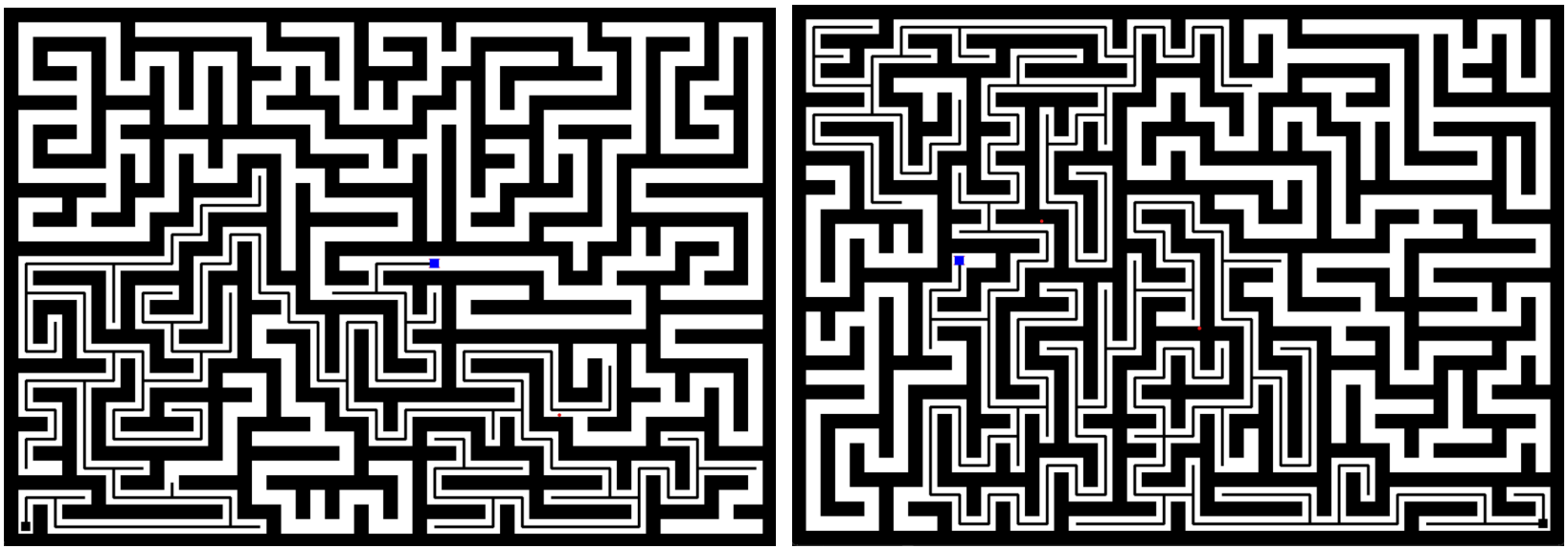


Figure 7: Labirinti risolti con policy appresa tramite l'algoritmo PAL. Dimensione labirinto: 53x37, algoritmo allenato su 50 environment con learning rate pari a 0.1 per Monte Carlo.

Conclusioni

Conclusioni

La policy derivata dall'algoritmo **PAL**, dove il modello adotta un approccio più aggressivo, porta a **curve di apprendimento non uniformi** caratterizzate da significative fluttuazioni della policy. Questo poiché i **parametri del modello cambiano drasticamente** da un'iterazione all'altra, e la **policy deve adattarsi di conseguenza**.

Una selezione oculata della policy derivata dall'algoritmo PAL può produrre risultati soddisfacenti nella risoluzione dei labirinti. Tuttavia, determinare un criterio di arresto adeguato basato sulla curva di apprendimento è difficile a causa dei rapidi cambiamenti tra le iterazioni.

Conclusioni

In contrasto, in **MAL**, dove la policy agisce come follower e i parametri del modello tendono verso un approccio più conservativo, la **convergenza è raggiunta più facilmente**. Esplorare inizializzazioni dei parametri diverse potrebbe essere interessante per guidare la policy verso una convergenza distintiva.

Questi risultati sono coerenti con i risultati presentati in [2].

Limitazioni e Future Works

Q-learning può derivare la policy ottimale indipendentemente dalla policy utilizzata per raccogliere i dati. Pertanto, è possibile salvare gli episodi eseguiti in un buffer esterno e utilizzarli ad ogni iterazione, combinandoli con nuovi episodi ottenuti eseguendo la politica aggiornata nell'ambiente.

La policy è in grado di apprendere regole solo su singoli stati, ma non su sequenze composte da coppie di azioni e stati eseguiti in precedenza. Ciò potrebbe potenzialmente limitare le prestazioni degli algoritmi.

Riferimenti

Bibliografia

P. Huang, M. Xu, F. Fang, and D. Zhao.
Robust reinforcement learning as a stackelberg game via adaptively-regularized adversarial training, 2022.

A. Rajeswaran, I. Mordatch, and V. Kumar.
A game theoretic framework for model based reinforcement learning, 2021.

R. S. Sutton and A. G. Barto.
Reinforcement Learning: An Introduction.
The MIT Press, second edition, 2018.

S. Tadelis.
Game theory: an introduction.
Princeton university press, 2013.

C. J. Watkins and P. Dayan.
Q-learning.
Machine learning, 8:279–292, 1992.

D. B. Wilson.
Generating random spanning trees more quickly than the cover time.
In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, page 296–303. Association for Computing Machinery, 1996.

Librerie

Matrix MDP gym, Paul Festor

`https://github.com/Paul-543NA/matrix-mdp-gym`

Maze Generator, Kalle Saariaho

`https://github.com/zeque92/RealTime3DwithPython`

Grazie per l'attenzione