3.3 PROGRAMMAZIONE DINAMICA

La *Programmazione Dinamica* (DP) fu introdotta da *Bellman* nel 1957 (Bellman, 1957) e consiste in una tecnica che si basa sulla suddivisione del problema in sotto-problemi e sull'utilizzo di sotto-strutture ottimali.

Nel *Reinforcement Learning*, la DP si riferisce ad una serie di algoritmi in grado di calcolare *policy ottimali* avendo a disposizione il modello completo dell'ambiente di un MDP. Nella DP vengono risolte in maniera iterativa le equazioni di *Bellman* con l'obiettivo di migliorare le approssimazioni delle *Value Functions*.

E' importante sottolineare che nel *Reinforcement Learning* esistono due tipologie di *task*:

- data una *policy* π nella *Prediction* l'obiettivo è quello di calcolare il valore delle *Value Functions*.
- Date alcune *policy* nel *Control* viene individuata la *policy ottimale* π_*

Di seguito vengono mostrate le componenti necessarie per l'utilizzo di un algoritmo di DP, chiamato *Policy Iteration*.

3.3.1 Policy Evaluation (Prediction)

Data una *policy* π, questa prima componente classificata come *Prediction* ha l'obiettivo di calcolare la *State-Value Function* per ogni stato. Formalmente può essere definita come segue:

$$\texttt{calcola } v_\pi(s), \quad \forall s \in S \tag{3.20}$$

Per calcolare questi valori viene utilizzata la seguente formula chiamata *Iterative Policy Evaluation*:

$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(\mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a v_k(s')) \tag{3.21}$$

3.3.2 Policy Improvement

La seconda componente ha invece l'obiettivo di migliorare la *policy* seguendo una strategia *greedy* e per farlo si basa sul teorema noto come *Policy Improvement Theorem*:

$$\forall s \in S, \quad q_\pi(s, \pi'(s)) \geqslant v_\pi(s) \Longrightarrow v_{\pi'} \geqslant v_\pi(s) \tag{3.22}$$

Questo permette di individuare una nuova *policy* applicando:

$$\pi'(s) = \arg\max_a \mathcal{R}_s^a + \gamma \sum_{s' \in S} \mathcal{P}_{ss'}^a \, v_\pi(s') \tag{3.23}$$

3.3.3 Policy Iteration

Data una *policy* π, le componenti appena viste permettono di individuare $v_\pi(s)$ utilizzando la *Policy Evaluation* e successivamente, data $v_\pi(s)$, di individuare una *policy* migliore attraverso la *Policy Improvement*.

La *Policy Iteration* si riferisce a una procedura che ha lo scopo di migliorare la *policy* combinando le componenti di *Evaluation* e *Improvement*. Nello specifico, come mostrato in Figura 3.2, le due componenti si alternano. Viene indicata con "E" la *Policy Evaluation* e con "I" la *Policy Improvement*:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Figura 3.2: Funzionamento *Policy Iteration* nella DP

3.3.4 Limiti Programmazione Dinamica

Seppure efficace per la risoluzione di problemi di RL, questa tecnica presenta dei limiti che ne rendono difficile l'utilizzo:

- risulta applicabile solamente se si conoscono le dinamiche complete del sistema, ovvero nel caso in cui si abbia a disposizione un modello perfetto dell'ambiente;
- richiede un'elevata potenza computazionale.

Tuttavia, si è ritenuto opportuno presentarla all'interno di questo elaborato in quanto costituisce la base ai diversi metodi impiegati per la risoluzione di un problema di RL, in cui si cerca di ottenere lo stesso risultato ad un costo computazionale inferiore e senza l'esigenza di utilizzare un modello dell'ambiente.