OTTIMIZZAZIONE RISK SENSITIVE PER IL FUNZIONAMENTO DI NODI SENSORE WIRELESS

RELATORE: Ch.mo Prof. Michele Rossi

LAUREANDO: Marco Rosada

A.A. 2012-2013

UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

OTTIMIZZAZIONE RISK SENSITIVE PER IL FUNZIONAMENTO DI NODI SENSORE WIRELESS

RELATORE: Ch.mo Prof. Michele Rossi

LAUREANDO: *Marco Rosada*

Padova, 08 ottobre 2013

Sommario

Gli algoritmi di apprendimento per rinforzo sono molto utili per massimizzare le performance di un dispositivo. Negli ultimi anni si è cercato di perfezionare questi algoritmi rendendoli *risk-sensitive*. In questo lavoro, si vuole applicare un simile approccio al fine di ottimizzare il funzionamento di un nodo sensore wireless, mantenendo sotto controllo il livello di carica della batteria del sensore stesso. In seguito ai recenti progressi tecnologici, è lecito assumere che il sensore sia alimentato da una fonte di energia rinnovabile, nella fattispecie un pannello fotovoltaico. Il ciclo di vita del sensore viene descritto da un processo decisionale di Markov, a cui viene applicato l'algoritmo studiato. I risultati ottenuti mediante un ottimizzatore scritto in linguaggio C vengono quindi analizzati per verificare la validità del metodo proposto.

Indice

Elenco delle figure

Elenco degli algoritmi

Capitolo 1

Introduzione

Il rischio è un concetto che fa parte della vita di tutti i giorni. Raramente si presenta l'opportunità di guadagnare qualcosa senza correre il rischio di rimetterci qualcos'altro. Anche tentando di mantenere lo status quo, fattori esterni al di fuori del nostro controllo possono avere un'influenza negativa o positiva sulla nostra situazione.

Analogamente, il rischio gioca un ruolo fondamentale nel *machine learning*. L'apprendimento automatico ha generalmente come scopo derivare una politica decisionale che massimizzi le performance di una macchina, o di un sistema. Queste macchine, come gli esseri umani, sono soggetti ai rischi e ai pericoli facenti parte dell'ambiente in cui si trovano. Per questo motivo, negli ultimi anni, sono stati proposti diversi metodi per rendere l'apprendimento automatico sensibile al rischio, ossia capaci di massimizzare le performance evitando contemporaneamente di incorrere in rischi eccessivi. In particolare, questa tesi si concentra sugli algoritmi di apprendimento per rinforzo, o *reinforcement learning*, nei quali un agente apprende in maniera del tutto autonomo, senza basarsi cioè su conoscenze pregresse ottenute da un soggetto esterno. L'unica elemento disponibile all'agente è un feedback ricevuto dall'ambiente circostante, che può essere visto come un'indicazione ricevuta sulla bontà delle scelte fatte da parte dell'agente.

Lo scopo di questo lavoro è integrare un algoritmo di apprendimento per rinforzo in modo da renderlo risk-sensitive. Per testarlo, è stato applicato ad un problema di ottimizzazione di interesse recente. Si vuole ottimizzare il funzionamento di un nodo sensore wireless, limitando il rischio di scaricare la batteria del sensore oltre una certa soglia.

Un sensore wireless è un dispositivo di dimensioni molto ridotte, con limitate capacità computazionali e di memorizzazione, a basso consumo di potenza e con un basso costo di produzione. Tipicamente è dotato di un sensore capace di rilevare e misurare determinate grandezze fisiche d'interesse nell'ambiente circostante, di una radio per la trasmissione e ricezione dei dati, di un'unità di controllo. Generalmente un sensore fa parte di una rete di altri sensori, che in questo caso prendono il nome di nodi. Questo tipo di reti, chiamate *wireless sensor network*, trovano applicazione nei più svariati campi. In origine, le prime applicazioni delle WSN furono in ambito militare, ma successivamente il loro utilizzo è stato esteso a monitoraggio ambientale, controllo del traffico, domotica e altri ambiti.

La maggior limitazione nell'utilizzo di questa tecnologia è dato dalla batteria dei sensori. Ogni sensore infatti è dotato di una batteria, che una volta scaricata sancisce la fine del ciclo di vita del sensore. Nel corso degli anni sono stati adottati diversi accorgimenti per ridurre al minimo il consumo energetico di un sensore, tuttavia senza poter eliminare il problema alla radice. Tuttavia, grazie a recenti progressi tecnologici è possibile dotare un sensore di un piccolo pannello fotovoltaico, rendendo possibile la ricarica della batteria. In questo modo, il ciclo di vita di un sensore diventa teoricamente infinito.

Va comunque notato che, data la grande influenza delle condizioni meteorologiche sul rendimento dei pannelli fotovoltaici, massimizzare il risparmio energetico è sempre uno degli obiettivi primari. Per questo motivo, ogni sensore ha la possibilità di comprimere i dati da inviare, in modo da poter risparmiare una parte dell'energia destinata alla trasmissione, a discapito dell'accuratezza del segnale ricevuto dal destinatario.

Nel nostro algoritmo di apprendimento per rinforzo quindi, la quantità da ottimizzare sarà in funzione del throughput del sensore e dell'accuratezza del segnale ricevuto dal destinatario, vincolata, come già accennato, alla soddisfazione di una condizione sul livello della batteria.

1.1 Struttura della tesi

Il *capitolo 2* della tesi è dedicato al reinforcement learning. Come prima cosa, ne vengono discussi i principi cardine e i fondamenti matematici. Dopodiché viene

fornita una panoramica dettagliata sui vari tipi di approcci risk-sensitive presenti in letteratura.

Il *capitolo 3* invece, riporta il modello matematico di un sensore wireless collegato ad una fonte di energia rinnovabile. Questo modello verrà poi utilizzato per testare l'algoritmo proposto.

Il *capitolo 4* contiene la descrizione dell'algoritmo di apprendimento per rinforzo risk-sensitive proposto.

Nel *capitolo 5* infine, vengono discussi i risultati ottenuti da un ottimizzatore scritto in C, che implementa l'algoritmo descritto nel capitolo 4e lo applica al modello proposto nel capitolo 3. Viene messa in luce infine l'incidenza della modifica dei vari parametri sulle politiche ottenute.

Capitolo 2

Risk-sensitive Reinforcement Learning

Con apprendimento per rinforzo, o *reinforcement learning*, si intende un processo di apprendimento basato sull'interazione con l'ambiente circostante. Volendo fare un paragone con la vita di tutti i giorni, si potrebbe pensare ad un bambino che gioca senza la supervisione di un insegnante. Spostandosi e agendo autonomamente, il bambino accumula una serie di informazioni con le quali può stabilire delle relazioni di causa-effetto, e quali azioni è meglio intraprendere per raggiungere un obiettivo. L'indipendenza del bambino è ciò che rende particolarmente interessante questo tipo di approccio. In particolare, nell'apprendimento per rinforzo, per ogni decisione presa, un agente riceve dall'ambiente un feedback, chiamato *reward*, che altro non è che una misura della bontà della scelta appena fatta. L'obiettivo dell'agente è massimizzare in ogni situazione il feedback ricevuto. Nel corso degli ultimi anni, si è cercato di introdurre un ulteriore elemento di condizionamento, il rischio. Come nel mondo reale, ogni azione comporta delle conseguenze, che in alcuni casi possono essere semplicemente indesiderate e in altri addirittura catastrofiche. Rendendo l'agente sensibile al rischio (*risk-sensitive*), si vuole evitare che l'avidità di conoscenza non comprometta l'intero processo di apprendimento. In questo capitolo verranno illustrati i principi cardine dell'apprendimento per rinforzo, e successivamente verrà fornita una panoramica sulle più importanti tecniche risk-sensitive.

2.1 Reinforcement Learning

L'apprendimento automatico, meglio noto come *Machine Learning*, è senza dubbio uno dei campi principali dell'intelligenza artificiale. L'idea di rendere una macchina o un calcolatore capace di apprendere ha da sempre rappresentato per la comunità scientifica una sfida interessante ed affascinante. I metodi proposti nel corso degli anni possono essere grossomodo suddivisi in tre categorie: algoritmi di apprendimento supervisionato, algoritmi di apprendimento non-supervisionato, e algoritmi di apprendimento per rinforzo. I primi forniscono all'agente una serie di esempi positivi e negativi. Attraverso queste coppie di input/output, l'agente accumula un'esperienza sufficiente a determinare autonomamente un proprio modello decisionale. Nell'apprendimento non-supervisionato invece, l'agente ha a disposizione solamente dei dati di input, che cerca di raggruppare in categorie rappresentative, ottenendo delle previsioni sugli input successivi. Un esempio di questo tipo di apprendimento è dato dagli algoritmi dei motori di ricerca, i quali, data una o più parole chiave, sono in grado di creare una lista di link rimandanti alle pagine che l'algoritmo ritiene attinenti alla ricerca effettuata. Infine l'apprendimento per rinforzo prevede che l'agente scopra la migliore azione da prendere in una data situazione provando ed osservando un reward ricevuto dall'ambiente. In molti casi, le azioni intraprese possono influenzare anche le situazioni e di conseguenza anche i reward successivi. Questi due aspetti (ricerca basata su interazione e *delayed* reward) sono le caratteristiche peculiari dell'apprendimento per rinforzo. Nel proseguo del capitolo forniremo una definizione dettagliata di questo paradigma di apprendimento.

2.1.1 Il modello Agente - Ambiente

L'apprendimento per rinforzo potrebbe essere definito come il problema di apprendere tramite interazione per raggiungere un obiettivo. Colui che apprende e che prende le decisioni viene chiamato *agente*. Tutto ciò con cui interagisce viene chiamato *ambiente* o *sistema*. Queste due componenti interagiscono continuamente, l'agente selezionando un'azione e l'ambiente rispondendo a queste azioni e presentando all'agente nuove situazioni. Inoltre, l'ambiente restituisce all'agente anche dei premi, chiamati *reward*, ossia dei valori numerici particolari che l'agente cerca di massimizzare. Una definizione completa e consistente del-

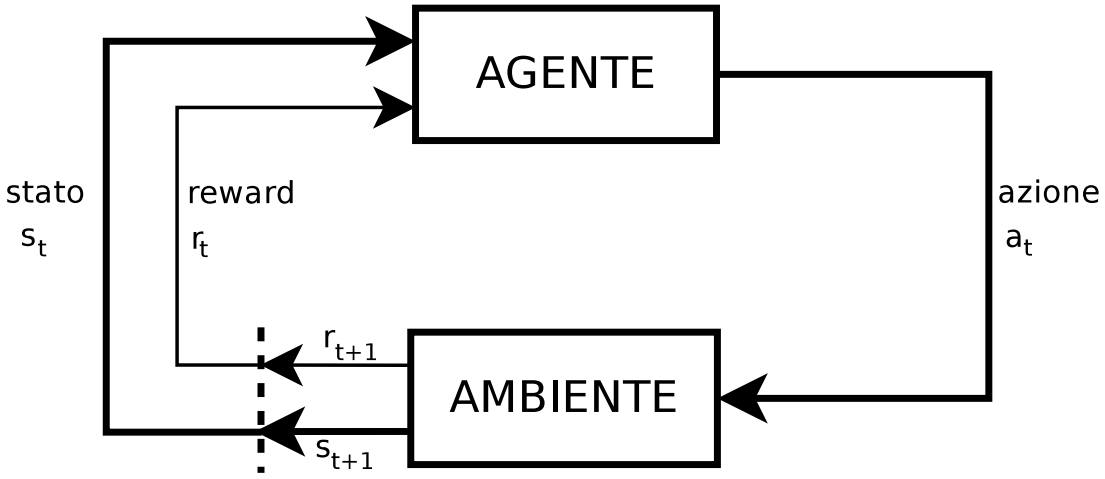

Figura 2.1: Modello agente-ambiente

l'ambiente rappresenta quindi un'istanza di un problema di apprendimento per rinforzo.

Più nel dettaglio, agente ed ambiente interagiscono ad intervalli di tempo discreti, $t = 0, 1, 2, 3, \ldots$. Ad ogni istante t, l'agente riceve una rappresentazione dello *stato* dell'ambiente, $s_t \in \mathcal{S}$, dove $\mathcal{S}$ è l'insieme degli stati possibili, e sulla base di questo sceglie un'*azione*, $a_t \in \mathcal{A}(s_t)$, dove $\mathcal{A}(s_t)$ è l'insieme delle azioni disponibili nello stato s_t. Al passo successivo, anche come conseguenza dell'azione presa, l'agente riceve un premio numerico, chiamato *reward*, $r_{t+1} \in \mathbb{R}$, e si ritrova in un nuovo stato, s_{t+1}. In figura 2.1 si può vedere un diagramma del modello di interazione tra agente ed ambiente.

Ad ogni passo, l'agente implementa una relazione tra stati e probabilità di scegliere ognuna della azioni possibili. Questa mappatura prende il nome di *politica*, o *policy*, e viene denotata con π, dove $\pi_t(s, a)$ è la probabilità che $a_t = a$ se $s_t = s$. In questa tesi ci occuperemo di *politiche deterministiche*, ovvero politiche dove $\pi_t(s, a) = 1$ per una sola delle azioni possibili nello stato s e $\pi_t(s, a) = 0$ per le rimanenti.

Un algoritmo di apprendimento per rinforzo descrive come l'agente cambia la propria politica in funzione dell'esperienza accumulata. Lo scopo dell'agente, è grossomodo massimizzare la quantità totale di reward che riceve dall'ambiente. E' bene notare che questo non significa massimizzare il reward immediato, ma il reward cumulativo nel lungo periodo.

2.1.2 Return

A questo punto possiamo definire formalmente cosa intendiamo con quantità totale di reward. Ipotizziamo che la sequenza di reward ricevuti a partire da un instante t sia denotata con $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$. Generalmente, ciò che vogliamo massimizzare è il *return medio*, o *expected return*, dove il ritorno, R_t, è definito come funzione della sequenza di reward. Nel caso più semplice il return è dato dalla somma dei reward:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T, \tag{2.1}$$

dove T è l'ultimo istante di tempo.

D'altro canto, spesso l'interazione tra agente e ambiente prosegue senza limiti temporali. In questi casi $T = \infty$, e di conseguenza il return, se espresso come semplice somma, tende anch'esso ad infinito. Per ovviare a questo problema si introduce un nuovo concetto, quello di *discounting*, o *sconto*. Seguendo questo approccio, l'agente cerca di selezionare azioni tali da massimizzare la somma dei reward scontati nel lungo periodo. Formalmente, sceglie a_t che massimizza l'*expected discounted return*:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

dove $\gamma \in [0, 1]$ è il *discount rate*, fattore di sconto. Il discount rate determina il valore effettivo dei reward futuri: un reward ricevuto k istanti temporali nel futuro vale solo γ^{k-1} volte quello che avrebbe contato se ricevuto immediatamente. Per $\gamma = 0$, l'agente è miope, e cerca di massimizzare solo i reward immediati. Se γ tende a 1, i reward futuri vengono presi maggiormente in considerazione, e l'agente diventa più lungimirante.

2.1.3 La proprietà di Markov

Si prenda in considerazione come un ambiente generico può rispondere, nell'istante $t+1$ all'azione presa dall'agente all'istante t. Tipicamente questa risposta potrebbe dipendere da tutto ciò che è avvenuto in precedenza. In questo caso la dinamica del sistema può essere definita esplicitando:

$$\Pr\left\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, t_1, s_0, a_0\right\}, \tag{2.3}$$

per ogni s', r, e tutti i valori possibili degli eventi passati: s_t, a_t, r_t,..., r_1,s_0,a_0. Tuttavia, se la dinamica dell'ambiente possiede la *proprietà di Markov*, la risposta del sistema all'istante $t+1$ dipende esclusivamente dalle rappresentazioni di stato e azione all'istante t. In questo caso, possiamo scrivere:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}, \tag{2.4}$$

per ogni s', r, s_t, e a_t.

Se un sistema possiede la proprietà di Markov, è possibile prevedere il prossimo stato e il prossimo reward medio dati solamente lo stato corrente e l'azione scelta. E' dimostrabile che, iterando questa equazione è possibile predirre tutti gli stati futuri e reward attesi con la sola conoscenza dello stato attuale come se si avesse a disposizione l'intera storia del sistema. Quindi, la miglior politica come funzione di uno stato di Markov è buona tanto quanto la miglior politica come funzione dell'intera storia. Appare a questo punto evidente come questa proprietà, se soddisfatta, semplifica notevolmente il problema dell'apprendimento per rinforzo.

2.1.4 Processi Decisionali di Markov

Un'istanza di apprendimento per rinforzo che soddisfi la proprietà di Markov viene detto *processo decisionale di Markov*(o *MDP, Markov Decision Process*). Se lo spazio degli stati e quello delle azioni sono finiti, prende il nome di processo decisionale di Markov finito.

Un MDP finito è definito da gli insiemi degli stati e delle azioni e dalla dinamica del sistema. Dato uno stato s ed un'azione a, la probabilità di ciascuno stato successivo, s', è:

$$\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}. \tag{2.5}$$

Queste quantità prendono il nome di *probabilità di transizione*. In maniera analoga, dato un qualsiasi stato corrente s, un'azione a e lo stato successivo s', il valore atteso del reward successivo è

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}. \tag{2.6}$$

Queste quantità, $\mathcal{P}^a_{ss'}$ e $\mathcal{R}^a_{ss'}$, descrivono esaustivamente gli aspetti principali della dinamica di un MDP finito. Nel proseguo della tesi, l'ambiente viene considerato un MDP finito.

2.1.5 Value Function

Quasi tutti gli algoritmi di apprendimento per rinforzo si basano sullo stimare delle *value function*, funzioni degli stati che stimano il return medio che è lecito aspettarsi. Ovviamente i reward che l'agente può immaginare di ricevere nel futuro dipendono dalle azioni che l'agente intraprenderà. Di conseguenza, una value function viene definita rispetto ad una politica particolare.

Il *value* di uno stato s sotto una politica π, denotato con $V^\pi(s)$, è il return atteso partendo da s e seguendo da lì in poi la politica π. Per un processo decisionale di Markov, $V^\pi(s)$ è uguale a:

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}, \tag{2.7}$$

dove $E_\pi\{\}$ denota il valore atteso dato che l'agente segue la politica π.

Una proprietà fondamentale delle value function è che esse soddisfano delle relazioni ricorsive particolari. Per una qualsiasi politica π e qualsiasi stato s, la seguente condizione è valida tra il value di s e il value dei suoi possibili successori:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t \middle| s_t = s\right\} \\ &= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s\right\} \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_{t+1} = s'\right\}\right] \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right], \end{aligned} \tag{2.8}$$

dove le azioni $a \in \mathcal{A}(s)$ e gli stati $s' \in \mathcal{S}$. L'equazione (2.8) è l'*equazione di Bellman per* V^π, ed esprime la relazione tra il value di uno stato e i value degli stati suoi successori. Il value di uno stato s deve essere uguale al return atteso dello stato successivo più il reward atteso tra i due stati.

La value function V^π è l'unica soluzione della relativa equazione di Bellman. Per questo motivo l'equazione (2.8) è l'elemento cardine di diversi algoritmi che calcolano, approssimano e apprendono V^π.

Value Function Ottime

Risolvere un'istanza di reinforcement learning significa grossomodo trovare una politica che raccolga molto reward nel lungo periodo. Per processi decisionali di Markov finiti, una politica può essere definita ottima nel modo seguente. Le value function definiscono un ordinamento parziale delle politiche. Una politica π è definita migliore o equivalente ad una politica π' se per ogni stato il suo return atteso è maggiore o uguale a quello di π'. In altre parole, $\pi \geq \pi'$ se e solo se $V^{\pi}(s) \geq V^{\pi'}(s)$ per ogni $s \in \mathcal{S}$. Esiste sempre una politica che è migliore o equivalente di tutte le altre. Questa prende il nome di *politica ottima*, o *optimal policy*. Siccome potrebbero essercene più di una, denotiamo tutte le politiche ottime con π^*. Queste condividono la stessa value function, chiamata *value function ottima* (*optimal state-value function*), V^*, pari a:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \tag{2.9}$$

per ogni $s \in \mathcal{S}$. Ovviamente essendo V^* la value function di una politica, deve soddisfare l'equazione di Bellman. Essendo ottima, la relativa equazione di Bellman può essere scritta senza fare riferimento ad una politica precisa. Per V^*, questa equazione prende il nome di *equazione di Bellman ottimale*:

$$V^*(s) = \max_{a} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \tag{2.10}$$

Per un MDP finito, l'equazione (2.10) ha un'unica soluzione. L'equazione (2.10) e di fatto un sistema di N equazioni in N incognite, dove $N = |\mathcal{S}|$ è il numero di stati possibili dell'ambiente. Se la dinamica del sistema ($\mathcal{P}^a_{ss'}$ e $\mathcal{R}^a_{ss'}$) è nota, allora il sistema di equazioni può essere risolto per V^*. Una volta calcolata V^*, trovare la politica ottima è relativamente semplice. Per ogni stato s, ci saranno una o più azioni per le quali il massimo nella equazioni di ottimalità di Bellman è raggiunto. Qualsiasi politica che assegna una probabilità diversa da zero solo a queste azioni è una politica ottima.

Sfortunatamente, nella maggior parte dei casi un agente può determinare la polita ottimale solo con costi computazionali estremamente alti. Gli aspetti critici del problema sono la quantità di computazione richiesta ad ogni singolo passo e la quantità di memoria necessaria per ottenere approssimazioni delle value function, delle politiche e del modello dell'ambiente stesso. Per questo motivo, specialmente

per modelli di un ambiente con un numero di stati molto elevato, gli algoritmi di apprendimento per rinforzo ottengono una approssimazione della politica ottima. Tuttavia la natura dinamica di questo paradigma di apprendimento, permette di ottenere politiche approssimate tali da raggiungere una grande accuratezza per stati molto frequenti, al prezzo di ottenere minore accuratezza per stati raramente visitati.

2.1.6 Algoritmi di Reinforcement Learning

I metodi per risolvere il problema dell'apprendimento per rinforzo possono essere divisi fondamentalmente in tre classi: *programmazione dinamica*, metodi *Monte Carlo*, e apprendimento basato su differenze temporali (*temporal difference learning*). Di seguito vengono presentate le caratteristiche principali di ciascuna. La trattazione esaustiva dei più importanti algoritmi delle diverse categorie è al di fuori dello scopo di questo lavoro. Per maggiori approfondimenti si rimanda a [Sutton and Barto, 1998].

Programmazione Dinamica

Il termine programmazione dinamica (DP) fa riferimento ad un insieme di algoritmi utilizzabili per trovare politiche ottime dato un modello perfetto dell'ambiente visto come processo decisionale di Markov (MDP).

L'idea di base è utilizzare la value function per organizzare la ricerca di buone politiche. Per farlo, vengono ciclicamente risolte le equazioni di Bellman (2.8) associate a ciascun stato, in modo da ottenere approssimazioni delle value function via via più accurate.

Questi algoritmi hanno sostanzialmente due punti deboli: primo, necessitano di conoscere completamente la dinamica del sistema in termini di probabilità di transizione tra gli stati, cosa non sempre possibile, secondo, hanno un costo computazionale elevato. Tuttavia, essi costituiscono la base teorica essenziale su cui sono stati costruiti tutti gli altri algoritmi di apprendimento per rinforzo, e di conseguenza rappresentano un tema di studio fondamentale.

I principali algoritmi di questa classe sono *Policy Iteration* e *Value Iteration*, i quali presentano diversi punti in comune e sono dal punto di vista della qualità dei risultati ottenibili praticamente equivalenti. Nel capitolo 4, Policy Iteration verrà descritto accuratamente, fornendone anche lo pseudo-codice.

Metodi Monte Carlo

I metodi Monte Carlo cercano di ottenere la politica ottima mediando dei return di esempio. E' quindi sufficiente un modello dell'ambiente in grado di generare queste transizioni di esempio. A differenza della programmazione dinamica, non è necessario conoscere la probabilità di tutte le transizioni possibili. In molti casi infatti è semplice generare dei campioni che soddisfino le distribuzioni di probabilità desiderate, mentre è impraticabile esprimere in modo esplicito la totalità delle distribuzioni di probabilità.

Questi algoritmi simulano una sequenza di esempio, chiamata *episodio*, e in base ai valori osservati aggiornano le stime dei value e le politiche. Iterando per un numero sufficiente di episodi i risultati ottenuti mostrano un accuratezza soddisfacente.

Rispetto agli algoritmi della classe precedente, gli algoritmi Monte Carlo non necessitano del modello completo del sistema. Tuttavia, offrono la possibilità di aggiornare value e politica solamente al termine di ogni simulazione, a differenza degli algoritmi di DP che aggiornano le stime ad ogni passo.

Temporal Difference Learning

Gli algoritmi di temporal difference learning (TD learning), nascono come combinazione di idee prese dalla programmazione dinamica e i metodi Monte Carlo. Come i metodi Monte Carlo, possono apprendere direttamente da degli episodi di esempio, senza un modello completo della dinamica del sistema. Come gli algoritmi di DP, aggiornano le stime basandosi in parte su altre stime apprese precedentemente, senza dover per forza attendere la fine dell'episodio.

Gli algoritmi TD più noti sono *Sarsa*([Rummery and Niranjan, 1994]) e *Q-Learning*([Watkins, 1992]).

2.2 Risk-sensitive RL

Come visto, generalmente gli algoritmi di apprendimento per rinforzo mirano a massimizzare il reward totale atteso di un processo decisionale di Markov. Negli ultimi vent'anni, diversi studi hanno segnalato i limiti di questa tecnica, particolarmente evidenti in casi in cui devono essere rispettate delle condizioni di sicurezza.

Per ovviare a queste problematiche sono state proposti diversi accorgimenti. La prima grande differenza tra le soluzioni proposte, è il significato attribuito alla parola *rischio* (o *risk*). Nella maggior parte dei casi, il rischio viene associato alla varianza del return atteso. In altre parole, l'obiettivo principale è prevenire che il return ottenuto sia molto più piccolo, o grande, del suo valore atteso (al crescere della varianza cresce anche il rischio, e vice versa). In altri studi invece, il rischio è visto come una misura associata agli stati dell'ambiente. In questo caso con rischio intendiamo la probabilità che, partendo da un certo stato, il MDP porti ad uno stato di errore. Ovviamente, entrambe le nozioni sono valide e presentano dei propri pro e contro.

Le soluzioni basate sulla prima definizione di rischio possono essere raggruppate in tre categorie differenti: *worst case control* ([Coraluppi and Marcus, 1999], [Heger, 1994a]), *exponential utility function* ([Koenig and Simmons, 1994], [Howard and Matheson, 1972]) e *expected value minus variance criterion* ([Filar et al., 1989]). Nel primo caso, la varianza è controllata massimizzando il peggior esito possibile. Nel secondo, il return è trasformato in modo da riflettere una misura soggettiva di utilità. Nell'ultimo, lo scopo è massimizzare il value atteso pesato con una certa frazione della varianza.

Dall'altro lato, solo pochi studi recenti hanno investigato la possibilità di integrare nell'apprendimento per rinforzo una nozione di rischio associato agli stati del sistema ([Geibel, 2001], [Fulkerson et al., 1998]). In questi lavori, l'idea di base è ottimizzare il return mantenendo contemporaneamente il rischio sotto una soglia prestabilita, pesando la value function con una *state-action-risk function*. Come si può immaginare, questo tipo di approccio è strettamente legato al *expected value minus variance criterion*.

2.2.1 Worst Case Control

Nel worst case control, con rischio intendiamo la probabilità che il return di una specifica istanza del processo sia decisamente peggiore del suo valore medio. Quindi, si vuole massimizzare la varianza del return, massimizzando il return del caso peggiore. Nelle prossime sezione, verrà presentato brevemente uno dei primi e più importanti lavori in questo campo ([Heger, 1994a]).

α-value Criterion

Il α-*value* del return di una politica π relativo ad uno stato s è definito come:

$$\hat{m}_\alpha = \hat{m}_\alpha \left(R^\pi \left(s\right)\right) =_{def} \min_r \left\{\Pr \left(R^\pi \left(s\right) < r\right) > \alpha\right\}, \tag{2.11}$$

dove $\alpha \in [0, 1)$ e $\Pr \left(R^\pi \left(s\right) < r\right)$ denota la probabilità che, se lo stato iniziale del sistema è s, il return sia inferiore a r. In [Heger, 1994b] è dimostrato che $\Pr \left(R^\pi \left(s\right) \geq \hat{m}_\alpha\right) \geq 1 - \alpha$. Ne consegue che un processo che inizia in uno stato s ha return maggiore o uguale a $\hat{m}_\alpha$ con probabilità almeno pari a $1 - \alpha$.

Secondo il α-value criterion, deve essere scelta una politica che massimizzi $\hat{m}_\alpha$.

Minimax Criterion

Il *minimax criterion* è un caso particolare del α-value criterion, dove $\alpha = 0$. Chiamiamo

$$\hat{V}^\pi(s) = \hat{m}_0 \left(R^\pi \left(s\right)\right) = \min_r \left\{\Pr \left(R^\pi \left(s\right) < r\right) > 0\right\} \tag{2.12}$$

il *value massimo* del costo di uno stato s (sotto la politica π). Il nome value massimo deriva dal fatto che

$$\Pr \left(R^\pi \left(s\right) \geq \hat{V}^\pi(s)\right) = 1$$

e

$$\Pr \left(R^\pi \left(s\right) > r\right) > 0$$

per ogni $r > \hat{V}^\pi \left(s\right)$. In altre parole, $\hat{V}^\pi \left(s\right)$ può essere visto come il peggior (minimo) return totale che può presentarsi se, partendo dallo stato s, si segue la politica π.

Al solito, la performance di una politica π viene misurata dalla value function $\hat{V}^\pi$. La value function ottima $\hat{V}^*$ è definita come

$$\forall s \in \mathcal{S} \qquad \hat{V}^*(s) = \max_\pi \hat{V}^\pi(s). \tag{2.13}$$

Una politica è ottima se la sua value function è uguale a $\hat{V}^*$.

La regola di aggiornamento delle politiche per il worst case control è quindi:

$$\pi \left(s\right) = \max_a \left(\mathcal{R}^a_{ss'} + \gamma \min_{s'} V \left(s'\right)\right). \tag{2.14}$$

Chiaramente, questa tecnica può essere utilizzata con i dovuti accorgimenti in qualsiasi algoritmo di apprendimento per rinforzo. E' bene tuttavia notare che il worst case control prende in considerazione sempre e solo il peggior evento possibile, anche se questo avviene con probabilità prossima allo zero. Spesso questo si rivela essere un atteggiamento eccessivamente conservativo e pessimista.

2.2.2 Utility Function

Un'altra strategia per controllare la varianza del return fa uso della *utility theory*[1], un sotto-campo della *decision theory*. Sostanzialmente, in questo caso il rischio è rappresentato da una *utility function* che assegna un numero reale, chiamato *utility*, ad ogni return possibile. L'idea di fondo p scegliere una mappatura regolabile adeguata tra reward e utility (*utility function*). Quindi, invece che massimizzare il return atteso, lo scopo è massimizzare la utility totale attesa. Modificando i parametri della utility function scelta, è possibile ottenere politiche con rischio inferiore o uguale ad un certo livello. La discussione è basata principalmente su [Howard and Matheson, 1972] e [Koenig and Simmons, 1994].

Una Classe Particolare di Utility Function

Evidentemente, per ottenere risultati soddisfacenti, la cosa più importante è trovare una classe di utility function adeguata. Infatti, nella maggior parte dei casi, per una funzione $u(R)$ e due return R_1 e R_2, $u(R_1 + R_2) \neq u(R_1) + u(R_2)$. In questa situazione, la proprietà di Markov non viene rispettata, e di conseguenza i metodi classici di apprendimento per rinforzo non sono utilizzabili.

Fortunatamente, per alcune classi di funzioni questo discorso non è valido.

Convex Exponential Utility Function. Introduciamo un nuovo concetto:

Definizione 2.1 (Certainty Equivalent)**.** *Per una utility function u,se l'utilità attesa è x, allora* $u^{-1}(x)$ *è chiamato il suo certainty equivalent. Rappresenta il return certo che siamo disposti ad accettare al posto di quello del MDP originale.*

In [Howard and Matheson, 1972] è stato dimostrato che per mantenere la proprietà di Markov una utility function deve soddisfare la seguente condizione:

[1] branca dell' economia, per una trattazione approfondita si veda: Fishburn, Peter C.. Utility Theory for Decision Making. Wiley, New York. 1970

Definizione 2.2 (Delta Property)**.** *Se tutti i reward sono aumentati di un valore arbitrario, allora il certainty equivalent è aumentato dello stesso valore.*

D'ora in poi, ci concentreremo su un gruppo ristretto di funzioni che soddisfano la *delta property*, chiamate *convex exponential function*

$$u(r) = \gamma^r, \tag{2.15}$$

dove $\gamma > 1$. Il valore di γ influenza la sensibilità al rischio dell'agente. Al crescere di γ, l'agente sarà più disposto rischiare, e viceversa.

A questo punto possiamo formalizzare la quantità che deve essere massimizzata, l'utility totale attesa di una politica.

Expected Total Utility. Data una politica π, la sua utility totale attesa corrisponde a $u[s]$, dove s è lo stato iniziale. La utility attesa per uno stato generico s è $u[s] = u(R(s)) = \gamma^{R(s)}$, dove $R(s)$ è il return ottenuto partendo dallo stato s. Dopo aver preso un' azione a, l'agente incorre, per ogni $s' \in \mathcal{S}$ in un reward $\mathcal{R}^a_{ss'}$ con probabilità $\mathcal{P}^a_{ss'}$. Dopodiché il processo si rinnova. Infatti, in s', la expected total utility sarà $u[s']$ e il certainty equivalent sarà $u^{-1}(u[s']) = \log_\gamma u[s']$. In base agli assiomi della utility theory, la utility totale attesa di s' può essere rimpiazzata da il suo certainty equivalent. Quindi l'agente avrà un return totale pari a $\mathcal{R}^a_{ss'} + u^{-1}(u[s'])$ con probabilità $\mathcal{P}^a_{ss'}$. La expected total utility di uno stato s può essere calcolata nel modo seguente:

$$\begin{aligned} u[s] &= \sum_{s' \in S} \mathcal{P}^a_{ss'}\, u\left(\mathcal{R}^a_{ss'} + u^{-1}(u[s'])\right) \\ &= \sum_{s' \in S} \mathcal{P}^a_{ss'}\, \gamma^{\mathcal{R}^a_{ss'} + u^{-1}(u[s'])} \\ &= \sum_{s' \in S} \mathcal{P}^a_{ss'}\, \gamma^{\mathcal{R}^a_{ss'}}\, \gamma^{u^{-1}(u[s'])} \\ &= \sum_{s' \in S} \mathcal{P}^a_{ss'}\, \gamma^{\mathcal{R}^a_{ss'}}\, u[s']\,. \end{aligned} \tag{2.16}$$

Questo sistema lineare ammette sempre un'unica soluzione.

Sostituendo questo sistema alle equazioni di Bellman ((2.8)), utilizzando un qualsiasi algoritmo noto di apprendimento per rinforzo è possibile ottenere delle politiche sensibili al rischio.

2.2.3 Rischio relativo agli stati

Il maggior svantaggio degli approcci basati sulla varianza del return atteso è che non si adattano bene a problemi dove l'occorrenza di stati di errore o indesiderati comporta conseguenze fatali per l'intero ambiente. Infatti,in [Geibel, 2001] è stato dimostrato empiricamente che una politica con una varianza piccola può avere un rischio molto grande (rischio inteso come probabilità di entrare in uno stato indesiderato). Questo implica che una politica che porta a stati di errore molto velocemente non per forza ha una varianza più alta di altre politiche più sicure. Quindi, per affrontare questo tipo di problemi di ottimizzazione, è necessario introdurre un nuovo concetto di rischio.

In [Geibel, 2001] (e inoltre in [Hans et al., 2008] e [Polo and Rebollo, 2011]), invece di minimizzare la varianza del return atteso, l'obiettivo è evitare di entrare in stati di errore o indesiderabili del processo decisionale di Markov. Al fine di calcolare una politica feasible con rischio vincolato, si cerca di trovare il compromesso ottimo tra return e risk, massimizzando il return atteso senza superare una certa soglia di rischio. Value e risk sono per questo pesati utilizzando un peso ξ per il value e -1 per il risk.

In questa sezione l'analisi del problema verrà limitata a MDP ad orizzonte finito, ovvero MDP in cui sono presenti degli stati, chiamati terminali, che se raggiunti determinano l'interruzione del processo stesso.

Definiamo il rischio come la probabilità che una sequenza di stati, generata seguendo una certa politica, termini in uno stato d'errore.

Definizione 2.3 (Risk.)**.** *Sia π una politica, sia $s_1, s_2, \ldots, s_k$ un sequenza di stati. Il risk dello stato s_1 è definito come*

$$\rho^{\pi}(s_1) = \Pr(\exists i \mid s_i \in \Phi), \tag{2.17}$$

dove s_i è uno stato della sequenza, Φ è l'insieme degli stati di errore (che sono anche stati terminali), e s_1 è lo stato iniziale della sequenza.

Minimizzazione del Rischio

Il rischio ρ^{π} può essere considerato una value function definita su un segnale di *costo* $\bar{r}$. Per semplificare la formulazione matematica, aumentiamo lo spazio degli

stati del processo decisionale di Markov con uno stato di assorbimento aggiuntivo η dove il sistema entra dopo aver raggiunto uno stato di Φ.

La funzione di costo $\bar{r}$ è definita da

$$\bar{r}^{\,a}_{s,s'} = \begin{cases} 1, & \text{se } s \in \Phi \text{ e } s' = \eta \\ 0, & \text{altrimenti} \end{cases} \tag{2.18}$$

Con questa costruzione una sequenza di stati, azioni e costi che inizia in un certo stato s se entra in uno stato di errore contiene esattamente una volta un costo $\bar{r} = 1$. Nel caso in cui il processo non entri mai in uno stato di errore, allora la sequenza dei costi $\bar{r}$ contiene solamente zeri. Pertanto, la probabilità che definisce il rischio può essere scritta come il valore atteso di un return cumulativo definito per la funzione di costo.

$$\rho^{\pi}(s) = E\left[\sum_{k=0}^{\infty} \bar{\gamma}^k \bar{r}_k\right], \tag{2.19}$$

dove $\bar{\gamma} = 1$ è il discount factor della funzione di costo e $\bar{r}_k$ sono i costi associati agli stati della sequenza. La dimostrazione può essere trovata in [Geibel, 2001].

Definiamo quindi la *state-risk function* come:

$$\begin{aligned} \bar{V}^{\pi}(s) &= \mathrm{E}\left[\bar{R}_0 + \bar{\gamma}\rho^{\pi}(s')\right] \\ &= \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{s,s'} \left[\, \bar{r}^{\,a}_{s,s'} + \bar{\gamma}\, \rho^{\pi}(s')\right]. \end{aligned} \tag{2.20}$$

Definizione del Problema

Definiamo la performance media di una politica π come:

$$V^{\pi} = \sum_s p_s V^{\pi}(s), \tag{2.21}$$

dove p_s è la probabilità di prendere s come stato iniziale. Consideriamo quindi il problema seguente:

$$\max_{\pi} V^{\pi} \qquad \text{tale che} \qquad \forall s \;:\; \rho^{\pi}(s) \leq \omega, \tag{2.22}$$

dove ω è il parametro che specifica il limite superiore desiderato per il rischio.

Pesare value e risk

Definiamo una nuova value function V_ξ^π, che è la somma pesata di value e risk, tale che

$$V_\xi^\pi(s) = \xi V^\pi(s) - \rho^\pi(s). \tag{2.23}$$

Il parametro $\xi \geq 0$ determina l'influenza dei valori V^π rispetto a quelli ρ^π.

Formalizzate le tre value function necessarie possiamo tracciare le linee guida di un algoritmo che risolva il problema (2.22). Anche in questo caso, con i relativi accorgimenti è possibile adattare questo approccio alle diverse classi di algoritmi di apprendimento per rinforzo.

Una volta stimate, in maniera diversa a seconda dell'algoritmo (ricordiamo le tre classi, DP, Monte Carlo e TD learning), $V^\pi(s)$ e $\rho^\pi(s)$, $V_\xi^\pi(s)$ viene calcolata come somma pesata ((2.23)). A questo punto è semplice modificare la politica π aumentandone le performance.

Per ξ costante, l'algoritmo cerca di calcolare una politica ottima π_ξ^* per la formulazione pesata ((2.23)). Tuttavia, aumentando progressivamente ξ, si riesce a dare alla value function V la massima influenza possibile. Inizialmente, ξ è impostato a zero, e l'output dell'algoritmo sarà una politica con rischio minimo π_0. Questa politica permette di capire se il problema vincolato ammette una soluzione. Dopodiché, il valore di ξ è aumentato passo dopo passo, fermandosi quando il risk di almeno uno stato supera la soglia ω. Così facendo si ottiene una politica ottima π_ξ^*. Per prevenire oscillazioni dell'algoritmo, è bene impostare i discount factor allo stesso valore $\bar{\gamma} = \gamma$. Di conseguenza, questo corrisponde ad utilizzare un rischio scontato definito come

$$\rho_\gamma^\pi(s) = E\left[\sum_{k=0}^{\infty} \gamma^k \, \bar{r}_k\right]. \tag{2.24}$$

Naturalmente, il rischio scontato (2.24) da maggiore peso a stati di errore incontrati nel futuro prossimo, in base al valore di γ.

2.2.4 Considerazioni sulle tecniche proposte

Come già accennato, è difficile stabilire quale dei tre approcci visti sia il migliore. Dovendo scegliere quale utilizzare, è bene in primo luogo capire quale concezione di rischio sia la più idonea al dominio di applicazione. Dal nostro punto di vista, il modo di intendere il rischio più naturale è correlarlo agli stati di un processo

Markoviano. In particolare, essendo lo scopo della tesi trovare delle politiche ottime che garantiscano che il livello di carica della batteria di un sensore wireless non scenda sotto una certa soglia, appare evidente come sia opportuno pensare il rischio come funzione dello stato della batteria stessa.

Capitolo 3

Modello del Sensore

Dopo aver illustrato le principali tecniche per risolvere problemi di ottimizzazione di politiche vincolati, introdurremo ora il modello matematico di un nodo sensore alimentato da una fonte di energia rinnovabile, che verrà utilizzato per testare l'algoritmo risk-sensitive oggetto di questa tesi. Come prima cosa varranno definite le varie componenti del sistema in termini qualitativi. Dopodiché verrà presentato il modello Markoviano dell'ambiente su cui testare l'ottimizzatore.

3.1 Il sistema

Come già accennato, il nodo sensore in analisi è collegato ad una fonte di energia rinnovabile. Di conseguenza, il sensore, avendo la possibilità di ricaricare la propria batteria, è potenzialmente sempre operativo. Il compito del sensore è quello di comunicare è monitorare un certo fenomeno e inviare le misurazioni raccolte ad un ricevente. Il sensore ha la possibilità di comprimere i dati da inviare. La compressione dà la possibilità al sensore di risparmiare un po' di energia per la trasmissione dei dati, al prezzo di avere una minore accuratezza sulla ricostruzione dei dati ad opera del ricevente. L' obbiettivo è trovare una politica decisionale che massimizzi contemporaneamente il throughput e l'accuratezza dei dati ricevuti, mantenendo il livello di carica della batteria sopra una soglia di guardia.

In figura 3.1 è riportato il modello del sistema appenda descritto, in cui è possibile individuare le seguenti componenti:

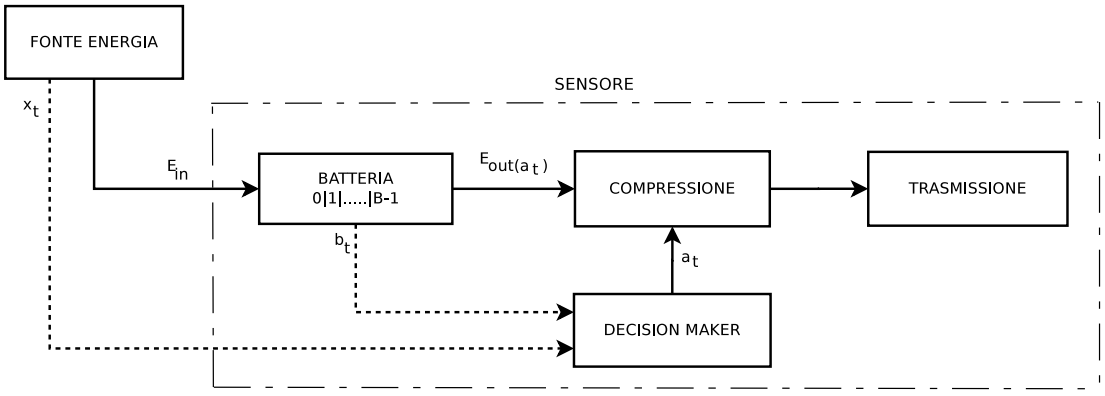

Figura 3.1: Schema del sistema

- *Fonte di energia rinnovabile:* è l'elemento che fornisce l'energia con cui è possibile ricaricare la batteria del sensore. Per motivi legati alla dimensione della sorgente in rapporto a quella del sensore, è auspicabile che la sorgente sia un pannello fotovoltaico. Ad ogni modo, in generale, si può utilizzare qualsiasi tipo di sorgente energetica.

- *Sensore:* alimentato dalla propria batteria interna, raccoglie i dati da inviare ad un ricevente. In base allo stato della batteria e della sorgente energetica esterna, sceglie il grado di compressione da utilizzare, comprime i dati e li invia.

- *Batteria:* fornisce al sensore l'energia necessaria per il suo funzionamento. L'unico requisito richiesto è che sia ricaricabile, e che possa quindi immagazzinare l'energia ricevuta dalla fonte rinnovabile. Il livello di carica della batteria viene aggiornato in base alle azioni intraprese dal sensore e dalla quantità di energia fornita dalla sorgente energetica in un dato istante.

Nel modello, l'energia richiesta e fornita alla batteria e quella in essa contenuta sono misurate in unità di carica.

3.2 Modello Markoviano

Il modello di un sistema è caratterizzato dallo spazio degli stati, quello delle azioni, le probabilità di transizione, e la funzione di reward. In un caso come il nostro, in cui sono presenti dei vincoli aggiuntivi di rischio, viene aggiunta anche una funzione di costo.

3.2.1 Stati

Nel nostro modello ogni stato è dato dallo stato della sorgente energetica e da quello della batteria. Sia $\mathcal{S}$ lo spazio degli stati, $\mathcal{X}$ lo spazio degli stati della fonte di energia rinnovabile e $\mathcal{B}$ lo spazio degli stati della batteria. Allora $\mathcal{S} = \mathcal{X} \times \mathcal{B}$.

Fonte Energetica

Come fonte di energia rinnovabile viene preso in considerazione un pannello fotovoltaico. Attualmente infatti è possibile reperire pannelli di questo tipo dalle dimensioni estremamente ridotte, che ben si adattano alle dimensioni altrettanto esigue di un sensore wireless. La dipendenza dalla quantità di luce emessa dal sole di questa sorgente, permette di dividere il funzionamento di un pannello fotovoltaico in due fasi ben distinte. Durante la notte infatti, data la totale assenza di luce solare, l'energia prodotta è costantemente pari a zero. Al contrario di giorno, l'energia prodotta varierà in base all'ora con una certa distribuzione.

Vengono identificati quindi due stati $x \in \mathcal{X} = \{0, 1\}$:

- *stato* $x = 0$ corrisponde alla notte, è lo stato identificato come *bad*. In questo stato l'energia prodotta E_{in} è pari a zero con probabilità uno.

$$\Pr\left(E_{in} = 0 | x = 0\right) = 1. \tag{3.1}$$

- *stato* $x = 1$. Identificato come *good*. Durante il giorno si assume che l'energia prodotta abbia una distribuzione gaussiana. Durante le ore centrali si registrano i valori massimi, con apice a mezzogiorno. Questo è dovuto alla traiettoria del sole, che nelle prime e nelle ultime ore della giornata è troppo bassa per garantire la radiazione necessaria al pannello fotovoltaico per generare una quantità di energia significativa. Ogni probabilità viene normalizzata in modo che la somma delle probabilità dia 1. Le unità di carica prodotte in questo caso quindi vengono distribuite su un intervallo $\{1, 2 \ldots, E_{max}\}$ secondo:

$$\Pr\left(E_{in} = i | x = 1\right) = \frac{\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(i-\mu)^2}{2\sigma^2}}}{\sum_{j=1}^{E_{max}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(j-\mu)^2}{2\sigma^2}}} \tag{3.2}$$

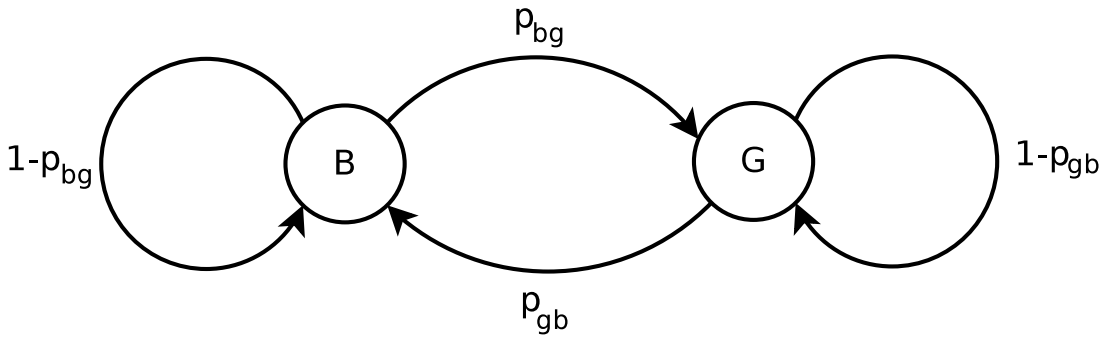

Figura 3.2: Modello della fonte energetica a due stati

Come evidenziato in figura 3.2, l' evoluzione dello stato della sorgente energetica è data da una catena di Markov a due stati la cui matrice di transizione è:

$$\Gamma = \begin{pmatrix} 1 - p_{bg} & p_{bg} \\ p_{gb} & 1 - {}_{gb} \end{pmatrix} \tag{3.3}$$

Il modello è stato scelto perché in grado di offrire un buon compromesso tra semplicità ed accuratezza. In maniera analoga è possibile ottenere un modello con numero di stati scelto in base al tipo di fonte di energia rinnovabile.

Batteria

La batteria può essere vista come un serbatoio di unità di carica, con capacità minima 0 e capacità massima $B - 1$. L'insieme degli stati del buffer energetico è $\mathcal{B} = \{0, 1, \ldots, B - 1\}$, ovvero l'insieme di tutti gli stati possibili di numero di unità di carica presenti nella batteria.

3.2.2 Azioni

In ogni istante decisionale, in base allo stato $s \in \mathcal{S}$ del sistema, viene scelta un'azione $a \in \mathcal{A}(s)$, dove $\mathcal{A}(s)$ è l'insieme delle azioni disponibili nello stato s. Nel nostro caso, l'azione corrisponde al grado di compressione dei dati, $a \in \{0, 0.1, 0.2, \ldots, 1\}$, per un numero totale di azioni pari a undici. In particolare, per $a = 0$, il nodo sensore rimane silenzioso, per $0 < a < 1$, i dati vengono compressi e poi inviati, e per $a = 1$ sui dati non viene fatta alcuna compressione e vengono inviati al ricevente direttamente.

Ogni azione comporta dei consumi energetici differenti, di conseguenza potrebbe accadere che in un certo istante l'energia richiesta per eseguire una certa

azione sia maggiore di quella disponibile nella batteria. In questo caso l'azione in analisi non sarà disponibile è lo spazio delle azioni per quello stato verrà ristretto.

Da un punto di vista formale, prendendo come riferimento [Zordan et al., 2012], l'energia richiesta per eseguire l'azione a nello stato s, denotata $E_{out}(s,a)$ è data dalla somma di due componenti. L'energia necessaria per effettuare la compressione dei dati, $E_c(a)$ e l'energia necessaria per la trasmissione dei dati, $E_{tx}(a)$. In particolare, l'energia richiesta per la compressione è funzione lineare del rapporto di compressione $\eta = a$:

$$E_c(a) = E_c(\eta) = \begin{cases} 0, & \eta = 0 \\ (\alpha\eta + \beta) N_b E_0, & 0 < \eta < 1 \\ 0, & \eta = 1 \end{cases} \tag{3.4}$$

dove N_b è il numero di bit da comprimere, E_0 è il consumo di energia per ogni ciclo della CPU del sensore, e α e β sono due coefficienti ottenuti empiricamente in [Zordan et al., 2012].

L'energia consumata per la trasmissione dei dati è funzione lineare del numero di bit da inviare e del rapporto di compressione η:

$$E_{tx}(\eta) = \eta N_b E'_{tx}, \tag{3.5}$$

dove N_b assume lo stesso significato visto in precedenza e E'_{tx} rappresenta l'energia richiesta per la trasmissione di un singolo bit.

L'energia richiesta quindi per eseguire un'azione a è data da:

$$E_{out}(s,a) = E_c(a) + E_{tx}(a). \tag{3.6}$$

In ogni stato $s = [x_s, b_s]$, l'insieme delle azioni disponibili $\mathcal{A}(s)$ è dato da tutte le azioni $a \in \mathcal{A}$ tali che $E_{out}(s,a) \leq b_s$.

3.2.3 Probabilità di transizione

Dopo aver formalizzato il processo di entrata (E_{in}) e quello di uscita (E_{out}) di energia dalla batteria del sensore, possiamo definire l'evoluzione del buffer energetico. Se il sistema, all'istante t, si trova nello stato $s_t = [x_t, b_t]$, e viene operata

l'azione $a \in \mathcal{A}(s)$, il valore del buffer energetico dello stato successivo s_{t+1} sarà:

$$b_{t+1} = \begin{cases} 0 & \text{se } b_t + E_{in} - E_{out}(s,a) <= 0 \\ B-1 & \text{se } b_t + E_{in} - E_{out}(s,a) >= B-1 \\ b_t + E_{in} - E_{out}(s,a) & \text{altrimenti} \end{cases} \tag{3.7}$$

La probabilità di transizione da uno stato $s_t = [x_t, b_t]$ allo stato $s_{t+1} = [x_{t+1}, b_{t+1}]$, data un'azione a_t, è zero se $a_t \notin \mathcal{A}(s_t)$, altrimenti ($a_t \in \mathcal{A}(s_t)$), è:

$$\Pr(s_{t+1}|s_t, a_t) = \Pr(E_{in}|x_t) \Pr(x_{t+1}|x_t) \tag{3.8}$$

dove $\Pr(x_{t+1}|x_t)$ può essere derivata dalla matrice (3.3). Per quanto riguarda invece $\Pr(E_{in}|x_t)$, il valore di E_{in} necessario per andare, data $a_t \in \mathcal{A}(s_t)$, da s_t a s_{t+1} può essere calcolato come segue:

$$\begin{cases} E_{in} \leq E_{out}(s_t, a_t) - b_t & \text{se } b_{t+1} = 0 \\ E_{in} \geq B - 1 + E_{out}(s_t, a_t) - b_t & \text{se } b_{t+1} = B-1 \\ E_{in} = B - 1 + E_{out}(s_t, a_t) - b_t & \text{altrimenti} \end{cases} \tag{3.9}$$

La relativa probabilità può dunque essere calcolata da (3.2).

3.2.4 Reward function

Ricordiamo che l'obiettivo del sensore è massimizzare il throughput e allo stesso tempo l'accuratezza del segnale ricostruito dal ricevente. In [Zordan et al., 2012] viene fatta un'analisi dettagliata di varie tecniche di compressione, al fine di determinare la migliore in termini di performance. I metodi che garantiscono il minor costo computazionale, e di conseguenza un maggior risparmio energetico,risultano essere metodi di compressione basati su approssimazione lineare, la cui funzione di errore relativo è stata calcolata come:

$$\xi(\eta) = \frac{p_1\eta^2 + p_2\eta + p3}{\eta + q_1}, \tag{3.10}$$

dove $\eta \in [0,1]$ è il rapporto di compressione e p_1, p_2, p_3 e q_1 sono parametri calcolati tramite numerical fitting.

La funzione di reward del modello è una funzione monotona crescente che vale zero se l'azione selezionata è $a = 0$, mentre tende a uno per $a = 1$.

$$r(a) = \begin{cases} 0 & a = 0 \\ 1 - \left(\frac{p_1a^2 + p_2a + p3}{a + q_1}\right) & 0 < a \leq 1 \end{cases} \tag{3.11}$$

3.2.5 Cost function

Il ruolo della funzione di costo è assegnare un reward negativo ad ogni stato della batteria. I livelli più bassi avranno costi più alti, mentre i livelli più alti avranno costi più bassi. Così facendo, durante l'apprendimento, oltre al reward precedentemente descritto, il sensore riceve un feedback sullo stato del buffer energetico, accumulando una conoscenza che renderà possibile scegliere una politica troppo dispendiosa in termini energetici.

La funzione di costo del modello è una funzione monotona decrescente, che vale uno per $b = 0$, e scende linearmente a zero fino a $b = b_{min}$, dove b_{min} rappresenta la soglia minima che si vuole considerare.

$$\bar{r}(s) = \begin{cases} 0 & b \geq b_{min} \\ \frac{b_{min} - b}{b_{min}} & b < b_{min} \end{cases} \tag{3.12}$$

3.3 Considerazioni sul modello

In questo capitolo è stato illustrato un processo decisionale di Markov (MDP) che ben rappresenta il ciclo di vita di un sensore wireless alimentato da una fonte di energia rinnovabile. Nel prossimo capitolo, verrà presentato un algoritmo di apprendimento di rinforzo risk-sensitive che applicato a questo modello restituisce una politica in grado di massimizzare il rendimento del nodo sensore senza che la batteria del nodo scenda sotto un certo livello scelto dall'utente.

Capitolo 4

Algoritmo

Sulla base di quanto riportato nei capitoli precedenti, è stato sviluppato un algoritmo di ottimizzazione in grado di agire su due fronti: da un lato, massimizzare il *return* atteso, dall'altro, mantenere il *risk* associato ad ogni stato del modello sotto una soglia prestabilita. In questo capitolo, verrà per prima cosa introdotto l'algoritmo di apprendimento per rinforzo di partenza, e, successivamente, in che modo è stato aggiornato in modo da renderlo risk-sensitive.

L'algoritmo si presenta come una variante di uno degli algoritmi di apprendimento per rinforzo più noti, *policy iteration*, i cui elementi principali vengono presentati nella prossima sezione.

4.1 Policy Iteration

Policy iteration fa parte della famiglia d algoritmi di *programmazione dinamica*, di conseguenza necessita di un modello perfetto dell'ambiente, inteso come Markov Decision Process (MDP). Gli svantaggi comportati da questo vincolo sono sostanzialmente due; primo, l'elevato costo computazionale limita l'applicabilità dell'algoritmo a modelli con numero di stati non elevato, secondo, non sempre è possibile determinare con esattezza tutte le probabilità di transizione tra gli stati di un modello. Per questi motivi, spesso vengono preferiti algoritmi basati su simulazioni real-time di episodi, come ad esempio *Q-Learning* e *Sarsa*.

D'altro canto è noto che la maggior parte degli algoritmi di apprendimento per rinforzo nascono con l'obbiettivo di simulare il comportamento di quelli di *programmazione dinamica* mantenendo un costo computazionale inferiore. Uti-

lizzare quindi l'algoritmo *policy iteration* come base di partenza è una scelta ideale nell'ottica di inglobare l'approccio risk sensitive qui proposto anche in altri algoritmi di apprendimento per rinforzo.

L'algoritmo consiste nell'iterazione di due fasi distinte, *policy evaluation*(1) e *policy improvement*(2). La prima, ha come obiettivo stabilire per ogni stato del modello il *return* atteso partendo da quello stato e seguendo la politica sotto esame. Per ogni stato, viene calcolata una stima della rispettiva *value function* tramite le equazioni di Bellman viste nel capitolo 2.

Algoritmo 1 Policy Evaluation

Input π, the policy to be evaluated
Initialize V(s)=0, for all $s \in S$
repeat
  $\Delta \leftarrow 0$
  for each $s \in S$: **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s'} \mathcal{P}_{s,s'}^{\pi(s)} \left[r_{s,s'}^{\pi(s)} + \gamma V(s') \right]$
    $\Delta \leftarrow \max(\Delta, \mid v - V(s))$
  end for
until $\Delta < \theta$ (a small positive number)

Da un punto di vista strettamente formale, l'algoritmo converge in un numero di iterazioni che tende ad infinito. Per questo motivo, viene fermato quando la stima ottenuta per ogni stato risulta essere sufficientemente vicina a quella ottenuta all'iterazione precedente.

Una volta calcolato il *return* medio della *policy*, è possibile determinare se, modificando la *policy* stessa, sia possibile ottenerne una più vicina all'ottimo. Ancora una volta, si ricorre all'utilizzo delle equazioni di Bellman. In particolare, per ogni stato del sistema si determina quale azione, presa nello stato in questione, massimizzi la somma tra il *reward* immediato ottenuto spostandosi in uno stato vicino e il *return* medio, calcolato durante la fase di *policy evaluation*, di questo vicino.

A questo punto, possiamo introdurre *policy iteration*(3) nella sua interezza. Dopo che una *policy* è stata migliorata sfruttando le stime tramite *policy evaluation*, viene valutata a sua volta ed eventualmente migliorata. Si ottiene quindi

Algoritmo 2 Policy Improvement

Input π, the policy to be improved
for each $s \in S$: **do**
  $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}^a_{s,s'} \left[r^a_{s,s'} + \gamma V(s') \right]$
end for

una sequenza monotona delle due fasi appena viste:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*,$$

dove $\xrightarrow{E}$ denota *policy evaluation* e $\xrightarrow{I}$ *policy improvement*. Dato che un MDP finito ha un numero finito di politiche possibili, l'algoritmo converge alla politica ottimale π^* e alla *value*-function ottimale V^* in un numero finito di iterazioni.

Algoritmo 3 Policy Iteration

Initialize $V(s) \in \mathbb{R}$ and $\pi \in \mathcal{A}(s)$ arbitrarily $\forall s \in S$
repeat
  Policy Evaluation
  Policy Improvement
until Policy is changing

Si noti che ogni *policy evaluation* parte utilizzando le stime dei *return* medi calcolate per la politica precedente. Questo generalmente comporta che la fase di *evaluation* converga in poche iterazioni, in quanto la value function cambia solo leggermente tra una politica e la successiva.

4.1.1 Prestazioni di Policy Iteration

Come tutti gli algoritmi di *Dynamic Programming*, *Policy Iteration* è difficilmente applicabile a problemi molto grandi, tuttavia tra tutti i metodi per risolvere MDP risulta essere il più efficiente. Se denotiamo con n e m il numero di stati e azioni, nel caso peggiore, il tempo necessario per trovare una *optimal policy* è polinomiale in n e m, nonostante il numero totale di *policy* possibili sia m^n. Quindi, *Policy Iteration* è esponenzialmente più veloce di qualsiasi algoritmo di ricerca esaustiva nello spazio delle *policy*. Per risolvere un MDP possono essere applicati anche metodi di *Programmazione Lineare*, che in alcuni casi al caso peggiore convergono

addirittura prima di *Policy Iteration*. Ad ogni modo, rispetto a *Policy Iteration*,i metodi di *Programmazione Lineare* diventano non applicabili per un numero di stati molto inferiore (di un fattore circa 100).

In sostanza, per problemi con un grande numero di stati, *Policy Iteration* e più in generale gli algoritmi di *Dynamic Programming* sono gli unici applicabili. Da un punto di vista pratico, questi algoritmi convergono molto più velocemente di quanto teoricamente previsto nel caso peggiore. E' inoltre opportuno notare che la grandezza di un problema rappresenta una difficoltà intrinseca al problema stesso e non del metodo di soluzione adottato. Per questo motivo, l'unica valida alternativa ai metodi di *Dynamic Programming* è rinunciare alla conoscenza completa del modello Markoviano del problema e basare l'apprendimento su dati empirici collezionati simulando più volte il processo Markoviano in analisi. Come visto in precedenza, questo tipo di approccio viene utilizzato dagli algoritmi di apprendimento facenti parte della classe *Temporal Difference*, come *Q-Learning* e *Sarsa*.

4.2 Risk-sensitive Policy Iteration

Come prima cosa, quando ci si trova ad affrontare un problema di minimizzazione del rischio, è bene definire con precisione cosa si intende con il concetto di *risk*. E' già stato visto come in letteratura a questo parola vengano attribuiti significati diversi a seconda del contesto applicativo. Riassumendo brevemente, è possibile suddividere questi approcci in due categorie: *risk* inteso come probabilità che il *return* sia di molto inferiore o superiore alla sua media, oppure *risk* inteso come probabilità che seguendo una certa *policy* il sistema si trovi in uno stato indesiderato o di errore. Entrambi sono evidentemente validi, e la scelta di quale adottare dipende esclusivamente da come viene definito il problema da risolvere. In questo lavoro, si è scelto di creare un algoritmo di ottimizzazione risk-sensitive partendo da un punto di vista del secondo tipo, dove il rischio è visto come funzione degli stati del sistema.

Risk

Nella nostra concezione, il rischio $\bar{R}^{\pi}(s)$ viene espresso da una *value function*, chiamata *risk function* , definita per un segnale di *costo* $\bar{r}(s)$. Ad ogni stato

stato del sistema viene associato un costo $\bar{r}(s)$; ovviamente gli stati maggiormente indesiderati avranno costo maggiore e quelli considerati non problematici costo inferiore. Il rischio associato ad uno stato s sarà quindi:

$$\bar{R}^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}\bar{r}_{t+k+1} \mid s_t = s\right\},$$

e la corrispondente equazione di Bellman per $\bar{R}^{\pi}(s)$ sarà:

$$\bar{R}^{\pi}(s) = \sum_{s'}\mathcal{P}_{s,s'}^{\pi(s)}\left[\bar{r}(s') + \gamma\bar{R}^{\pi}(s')\right].$$

dove $\gamma \in [0,1]$ rappresenta il *discount factor*. Con esclusivamente questi elementi, è possibile ottenere un algoritmo per minimizzare il rischio. Infatti, basta applicare l'algoritmo di *Policy Iteration* sostituendo la value function $V^{\pi}(s)$ con $T^{\pi}(s) = -\bar{R}^{\pi}(s)$.

Definizione del Problema

Nella maggior parte dei casi, l'obbiettivo non è minimizzare il rischio ma trovare piuttosto un compromesso tra *value* e *risk*. Nel nostro caso infatti, quello che si vuole ottenere è massimizzare il return mantenendo il rischio sotto una certa soglia, grande a piacere. In quest'ottica, viene stabilita la quantità di rischio ω che si è disposti ad accettare, con lo scopo di non penalizzare eccessivamente il *return* del sistema.

A questo punto, siamo in grado di definire il problema. Sia S' l'insieme dei possibili stati iniziali del sistema. Per uno stato $s \in S'$, sia p_s la probabilità di selezionare s come stato iniziale. Il valore di

$$V^{\pi} = \sum_{s\in S'} p_s V^{\pi}(s) \tag{4.1}$$

corrisponde alla performance media degli stati iniziali sistema. Nel caso in cui $S = S'$, (4.1) corrisponde al valore della policy π. Definiamo quindi il problema con vincoli:

$$\max_{\pi} V^{\pi} \qquad \text{such that } \forall s \in S',\ \bar{R}^{\pi}(s) \leq \omega. \tag{4.2}$$

Una policy che soddisfa (4.2) è detta *feasible*. In base al valore di ω, l'insieme delle *feasible policy* potrebbe essere vuoto. La miglior policy tra quelle *feasible* è la policy ottima per il problema (4.2).

Weighted Value-Risk Function

Per risolvere il problema (4.2), è stato ideato un algoritmo euristico. L'algoritmo si basa su intuizioni presenti in [Geibel, 2001] e [Fulkerson et al., 1998]. Introduciamo una nuova value function V_ξ^π, somma pesata di risk e value:

$$V_\xi^\pi(s) = \xi V^\pi(s) - \bar{R}^\pi(s). \tag{4.3}$$

Il fattore lagrangiano $\xi \geq 0$ determina il grado di influenza dei V^π-value rispetto ai $\bar{R}^\pi$-values. Per $\xi = 0$, V_ξ^π è pari all'opposto di $\bar{R}^\pi$, il che implica che una massimizzazione di V_0^π comporta una minimizzazione di $\bar{R}^\pi$. Per $\xi \to \infty$, la value function originale domina la misura di rischio.

Adattamento di ξ

Fissato un valore per il parametro ξ, l'algoritmo esegue la procedura di *Policy Iteration*, a cui vengono apportate alcune modifiche. Come prima cosa, invece di V^π, viene massimizzata la nuova value function (4.3). Secondo, durante la fase di *policy evaluation*, *value* e *risk* vengono valutati separatamente, in modo da ottenere una stima precisa per entrambi. Infatti, il numero di iterazioni interne a *policy evaluation* necessarie per convergere potrebbero essere molto diverso tra *value* e *risk*. Nel caso, valutare V_ξ^π in un'unica fase invece che separarne le componenti, porterebbe ad avere stime poco accurate. Dopo aver determinato la miglior policy, è necessario controllare che il vincolo sul *risk* sia rispettato per ogni stato. Infatti, nel caso in cui il parametro ξ fosse stato scelto in modo inadeguato, la policy restituita potrebbe non essere *feasible*. Lo pseudocodice dell'algoritmo di apprendimento con ξ costante è riportato in 4.

Algoritmo 4 Fixed ξ Risk-sensitive Policy Iteration

Initialize $V(s)$, $\bar{R}(s)$, $V_{\xi}^{\pi}(s)$ and $\pi \in \mathcal{A}(s)$ arbitrarily $\forall s \in S$
repeat
  repeat
    $\Delta \leftarrow 0$
    for each $s \in S$: **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \sum_{s'} \mathcal{P}_{s,s'}^{\pi(s)} \left[r_{s,s'}^{\pi(s)} + \gamma V(s') \right]$
      $\Delta \leftarrow \max(\Delta, \mid v - V(s))$
    end for
  until $\Delta < \theta$ (a small positive number)
  repeat
    $\Delta \leftarrow 0$
    for each $s \in S$: **do**
      $z \leftarrow \bar{R}(s)$
      $\bar{R}(s) \leftarrow \sum_{s'} \mathcal{P}_{s,s'}^{\pi(s)} \left[\bar{r}_{s,s'}^{\pi(s)} + \gamma \bar{R}(s') \right]$
      $\Delta \leftarrow \max\left(\Delta, \mid z - \bar{R}(s)\right)$
    end for
  until $\Delta < \theta$ (a small positive number)
  for each $s \in S$: **do**
    $V_{\xi}(s) = \xi V(s) - \bar{R}(s)$
  end for
  for each $s \in S$: **do**
    $\pi(s) \leftarrow \arg\max_a \left\{ \xi \sum_{s'} \mathcal{P}_{s,s'}^{a} \left[r_{s,s'}^{a} + \gamma V(s') \right] - \sum_{s'} \mathcal{P}_{s,s'}^{a} \left[\bar{r}_{s,s'}^{a} + \gamma \bar{R}(s') \right] \right\}$
  end for
until policy is changing
for each $s \in S$: **do**
  if $\bar{R}^{\pi}(s) \geq \omega$ **then**
    return no policy
  end if
end for
return π

La scelta del valore da assegnare al parametro ξ rappresenta evidentemente un elemento di criticità. Per questo motivo è opportuno modificare l'algoritmo in modo da adattare progressivamente il valore di ξ, facendo sì che i vincoli sul rischio non vengano violati.

Iniziando l'apprendimento, è opportuno settare $\xi = 0$. Così facendo saremo in grado di stabilire il rischio minimo di ogni stato del sistema, e se esiste almeno una policy *feasible*. Notiamo che il rischio minimo non dipende dalla politica adottata ma esclusivamente dalla funzione di costo e dal modello del sistema.

Successivamente l'algoritmo reitera, aumentando ξ di un certo ϵ, fino a quando il rischio $\bar{R}(s)$ di un qualsiasi stato s supera la soglia ω. Al crescere di ξ, i valori $V(s)$ assumono maggiore rilevanza rispetto ai valori di rischio $\bar{R}(s)$. A livello pratico questo si traduce nella selezione di azioni in grado di garantire value $V(s)$ maggiori, anche se questo generalmente comporta anche valori di rischio $\bar{R}(s)$ maggiori. Al fine di far convergere la fase di *Policy Iteration* per il valore successivo di ξ nel minor tempo possibile, i valori $V(s)$ e $\bar{R}(s)$ calcolati per ξ vengono usati come valori di inizializzazione per determinare la policy π per $\xi \leftarrow \xi + \epsilon$. Lo scopo di aumentare ξ è quello di dare alla value function V la massima incidenza possibile. Inoltre, questa tecnica risolve il problema della scelta del valore di ξ, che come abbiamo visto è determinante per ottenere risultati soddisfacenti.

Scelta della soglia di rischio ω

Un altro elemento di criticità da valutare è sicuramente la scelta del valore del rischio ω che si è disposti ad accettare per aumentare i valori V. Come visto in precedenza, il rischio associato ad ogni stato del sistema dipende dalla funzione di costo, facente parte del modello del sistema stesso. Dovendo l'algoritmo essere il più universale possibile e applicabile in contesti differenti, è opportuno che la soglia ω venga calcolata come funzione del rischio minimo associato ad uno stato.

$$\omega(s) = \mathrm{f}\left(\bar{R}_{\min}(s)\right), \tag{4.4}$$

dove $\bar{R}_{\min}(s)$ denota il rischio minimo associato ad uno stato s. In altre parole, correlando il parametro dell'algoritmo al modello del sistema, come per ξ, si elimina il problema di dover settare manualmente e in termini assoluti il valore di ω.

L'algoritmo quindi si comporterà come segue:

1. Per ogni stato s, calcola rischio minimo $\bar{R}_{\min}(s)$ $(\xi = 0)$
2. Per ogni stato s, calcola soglia $\omega(s) = (1 + \alpha)\,\bar{R}_{\min}(s)$

dove $\alpha \in [0, 1]$ indica l'aumento percentuale scelto.

Una volta stabilite le soglie di rischio $\omega(s)$, l'algoritmo apprende, per valori crescenti di ξ, la policy ottima π_ξ, terminando quando i vincoli sul rischio non vengono più rispettati.

Scelta del discount factor γ

Come noto, il discount factor $\gamma \in [0, 1]$ è il parametro di un algoritmo di programmazione dinamica che determina il grado di incidenza su una value function di *reward* futuri. Per $\gamma = 0$ verrà data rilevanza solamente ai *reward* immediati, ottenendo una politica più aggressiva. Per valori di γ prossimi ad 1, vengono presi in considerazione un numero crescente di *reward*, rendendo l'algoritmo più lungimirante.

Nel nostro caso, i discount factor sono due: uno per la value function tradizionale, e uno per la risk function. Tuttavia, per garantire la convergenza dell'algoritmo di apprendimento, è necessario che entrambe le funzioni adottino lo stesso discount factor $\gamma < 1$. Infatti, in questo caso, V_ξ può essere vista come una funzione standard con reward $\xi r - \bar{r}$, e di conseguenza converge.

Infine, l'ultimo parametro dell'algoritmo di apprendimento da esaminare è Δ, il quale determina il numero di iterazioni necessarie affinché le fasi di valutazione della policy convergano. Al crescere di Δ, il numero di iterazioni diminuirà, a discapito dell'accuratezza delle stime ottenute.

Di seguito (5) viene riportato lo pseudocodice dell'algoritmo completo.

Algoritmo 5 Risk-sensitive Policy Iteration

Initialize $V(s) \in \mathbb{R}$ and $\pi \in \mathcal{A}(s)$ arbitrarily $\forall s \in S$
Perform Fixed ξ Policy Iteration for $\xi = 0$
for each $s \in S$: **do**
  $\omega(s) \leftarrow (1+\alpha)\bar{R}(s)$
end for
repeat
  $\pi \leftarrow \pi_\xi$
  $\xi \leftarrow \xi + \epsilon$
  Perform Fixed ξ Policy Iteration for ξ
until $\exists s$ s.t. $\bar{R}(s) > \omega(s)$
return π

Capitolo 5

Risultati

Nei capitoli precedenti, dopo aver descritto un modello per la rappresentazione del funzionamento di un sensore wireless alimentato da un pannello fotovoltaico, è stato introdotto un algoritmo, ad esso applicabile, che consente di individuare la politica ottima per il dispositivo. In questo capitolo vengono illustrati i risultati ottenuti, analizzando il modo e la misura in cui i parametri del modello e dell'algoritmo influiscono su questi.

5.1 Parametri del Modello

Ai parametri del modello introdotto nel capitolo 3, sono stati assegnati i valori riportati di seguito, per la maggior parte sulla base di [Zordan et al., 2012]. Alcuni parametri invece, possono essere configurati a piacimento, per evidenziare alcuni aspetti rispetto ad altri. Ad esempio, per quanto riguarda il modello della fonte energetica, modificando i parametri è possibile adattarne il comportamento simulando diversi condizioni meteorologiche.

5.1.1 Consumi Energetici

Di seguito, i valori dei parametri del modello relativi ai consumi energetici per la compressione.

Parametro	Valore
α	$16,1$
β	$105,4$
N_b	960
E_0	$0,725$
E'_{tx}	230

dove N_b è espresso in bit, mentre E_0 e E'_{tx} in nano Joule (nJ).

La capienza del buffer energetico può essere modificata in base alle esigenze. I risultati esposti in questo capitolo si riferiscono ad un buffer di 1000 unità di carica, dove un'unità di carica è pari a 10 nJ.

5.1.2 Fonte Rinnovabile

Per quanto riguarda la fonte di energia rinnovabile, in modo da avere due cambi di stato nell'arco della giornata (da giorno a notte e viceversa), la probabilità di transizione è stata impostata come $p_{gb} = p_{bg} = 1/86400$. I valori invece di E_{max}, del valore atteso μ, entrambi espressi in unità di carica, e della varianza σ^2 possono essere adattati secondo gli scopi. I risultati esposti sono stati ottenuti per $E_{max} = 25$, $\mu = 12,5$, $\sigma^2 = 0.1$.

5.1.3 Reward Function

In base a quanto visto nel capitolo 3, la reward function è definita per i seguenti parametri:

Parametro	Valore
p_1	1.03
p_2	1.70
p_3	0.17
q_1	0.002

5.2 Analisi delle Politiche

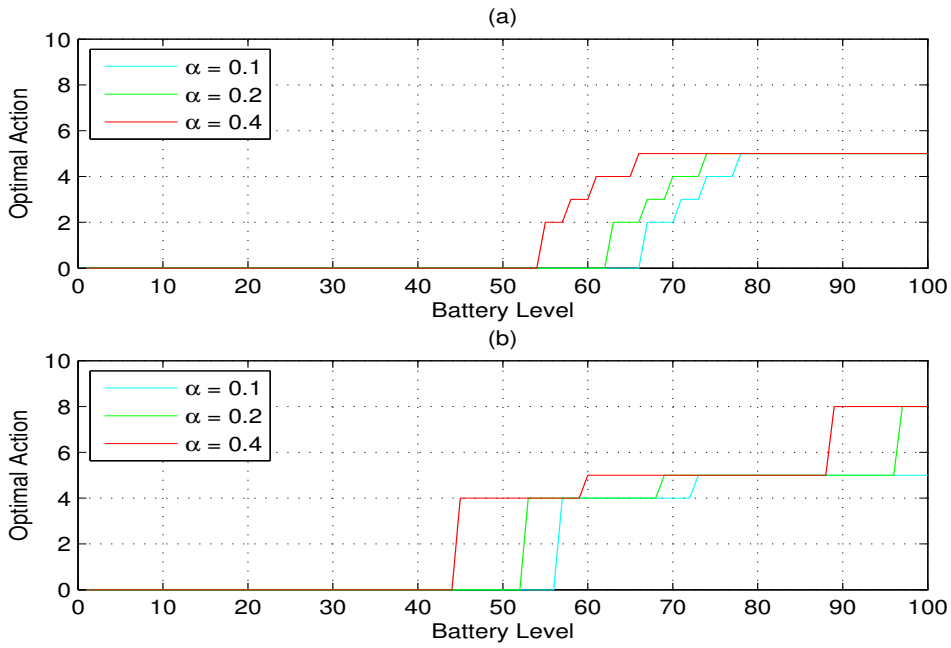

Figura 5.1: Politiche ottime al variare di α: (a) stato sorgente $x = 0$, (b) stato sorgente $x = 1$.

Dopo aver configurato adeguatamente i parametri del modello, è possibile utilizzare l'ottimizzatore per stabilire quali siano le politiche ottime per il funzionamento del sensore.

In figura 5.1, sono riportate le politiche ottenute al variare di α, il parametro dell'algoritmo che determina la quantità di rischio che si è disposti ad accettare (al crescere di α, aumenta la soglia di rischio tollerabile). Come si può notare, le politiche ottenute mostrano le caratteristiche che è lecito auspicare:

- le politiche sono sempre monotone crescenti, il che significa che maggiore è l'energia a disposizione del sensore e minore sarà la compressione effettuata sui dati, in modo da minimizzare l'errore relativo dei dati ricevuti dal destinatario. Viceversa, nel caso in cui il buffer energetico sia quasi vuoto, il sensore nel caso peggiore rimarrà silenzioso, oppure comprimerà pesantemente i dati da inviare, per risparmiare più energia possibile.

- il comportamento consigliato nelle ore diurne ($x = 1$) è più aggressivo che nelle ore notturne ($x = 0$). Ovviamente, avendo durante il giorno la pos-

sibilità di ricaricare la propria batteria, il sensore sarà più propenso a consumare energia per massimizzare il return. D'altro canto, durante la notte l'apporto energetico del pannello fotovoltaico è nullo, e un atteggiamento poco parsimonioso farebbe aumentare sensibilmente il rischio di svuotare completamente il buffer energetico.

- sia di notte che di giorno, al crescere di α, le politiche diventano più aggressive. Infatti, aumentando α, aumenta la soglia di rischio accettabile. Di conseguenza, il sensore comincia ad inviare dati a livelli di batteria più bassi e comprimendone meno. Dai grafici si nota come al crescere di α le politiche slittino verso il basso mantenendo lo stesso andamento.

Possiamo quindi affermare che i risultati ottenuti sono soddisfacenti.

5.3 Analisi delle Value Function

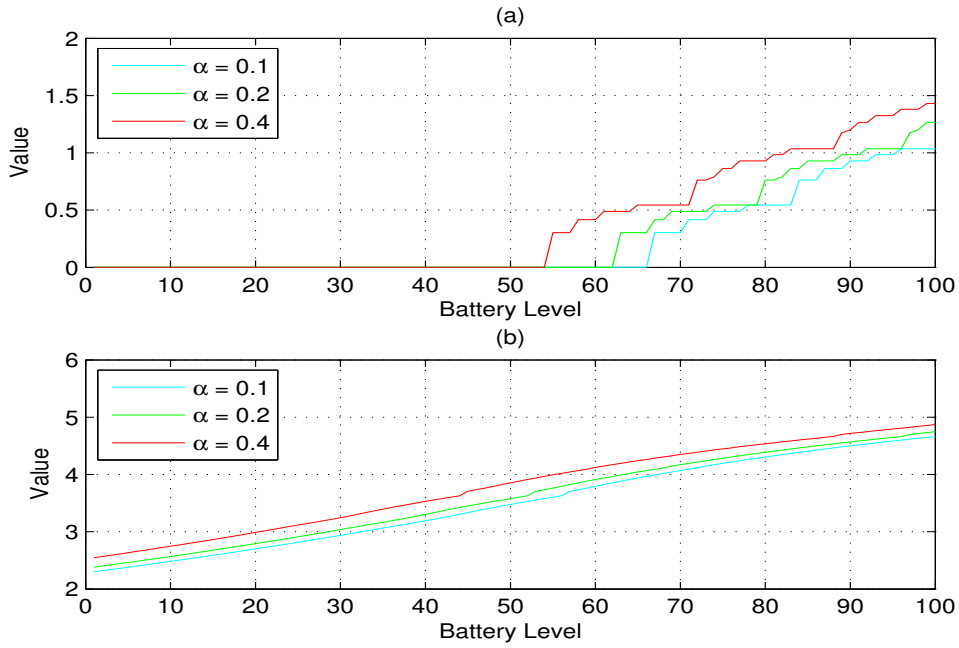

Figura 5.2: Value delle politiche ottime, al variare di α: (a) stato sorgente $x = 0$, (b) stato sorgente $x = 1$.

In questa sezione vengono analizzate le value function relative alle politiche ottime viste in precedenza.

In figura 5.2 sono riportati i return medi delle politiche, sia per quanto riguarda le ore diurne che quelle notturne.

Si può notare come, l'andamento è anche in questo caso monotono crescente. Infatti, per maggiore è l'energia a disposizione del sensore e maggiore è quella utilizzabile per l'invio dei dati.

I valori registrati aumentano al crescere di α. Anche in questo caso, è indicazione della bontà dell'algoritmo. Se così non fosse, l'incremento della soglia di rischio accettato non corrisponderebbe ad un incremento del return medio. In altre parole, il sensore si assumerebbe dei rischi inutili, senza aumentare le proprie performance.

Ancor più che nell'analisi riguardanti le politiche, appare evidente la netta differenza tra le due fasi della giornata. Il return medio durante il dì, è, come lecito aspettarsi, di gran lunga superiore a quello notturno.

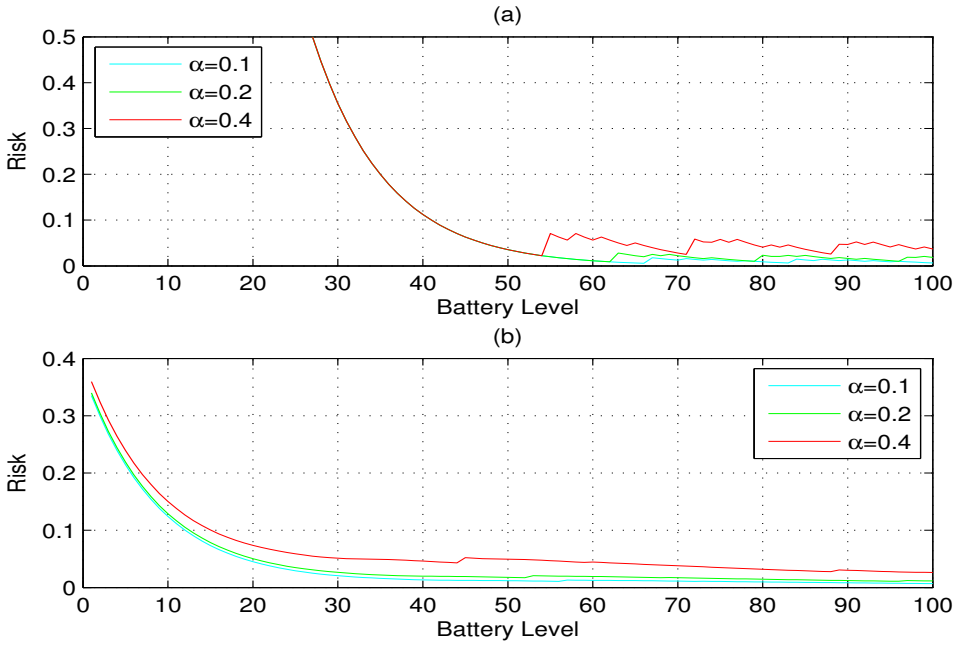

Figura 5.3: Rischio al variare di α: (a) stato sorgente $x = 0$, (b) stato sorgente $x = 1$.

Per quanto riguarda il rischio, in figura 5.3 sono riportati i costi delle varie politiche. Anche qui vengono confermate le supposizioni iniziali. Alzando il parametro α, l'agente diventa più aggressivo, aumentando il valore del rischio accumulato. In particolare, nel caso in cui la sorgente energetica sia nello stato

cattivo, il costo assume valori elevati, a causa della bassa probabilità di ricaricare la batteria in un lasso di tempo breve.

In figura 5.4, viene messo a confronto il rischio della politica ottenuta per $\alpha = 0.1$ e la relativa soglia. Si può vedere come il vincolo sul rischio sia sempre rispettato.

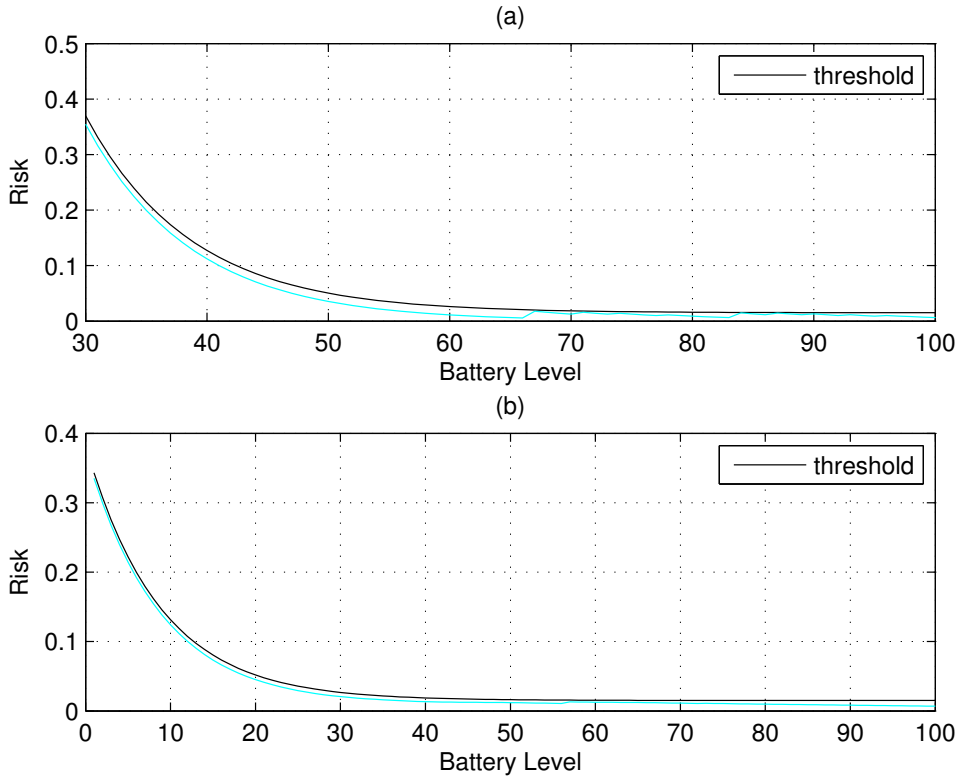

Figura 5.4: Rischio e soglia di rischio per $\alpha = 0.1$: (a) stato sorgente $x = 0$, (b) stato sorgente $x = 1$.

La funzione pesata di value e risk è infine riportata in figura 5.5. Come si può vedere, sono funzioni monotone crescenti, stretta conseguenza del fatto che è questa la funzione che viene massimizzata durante la fase di policy improvement dell'algoritmo.

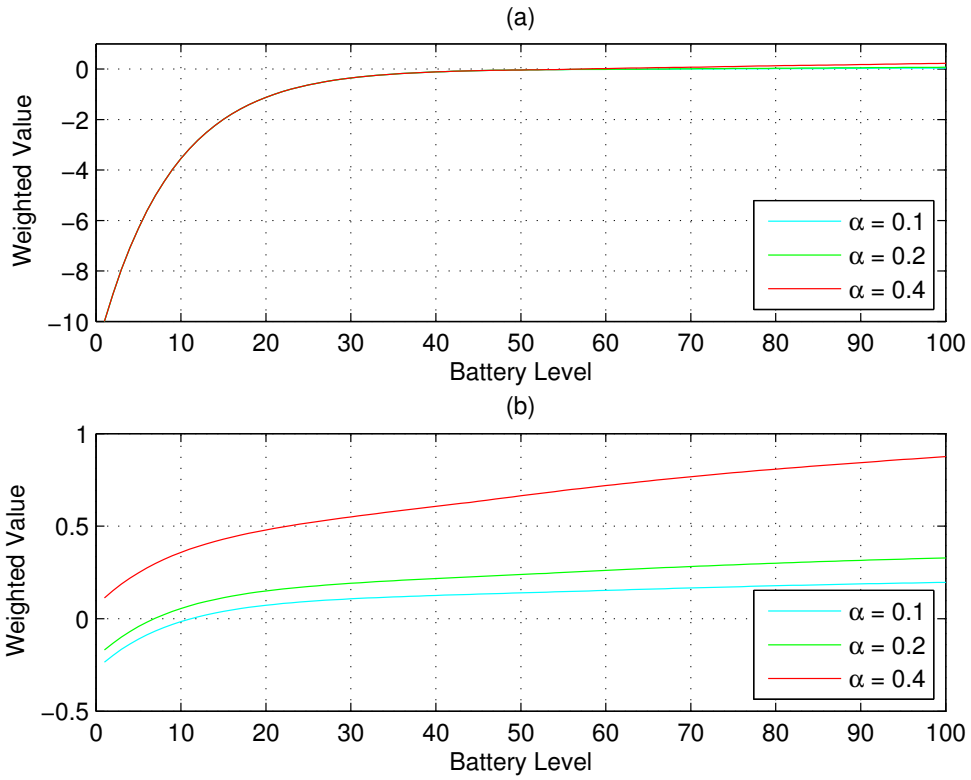

Figura 5.5: Value pesato col risk al variare di α: (a) stato sorgente $x = 0$, (b) stato sorgente $x = 1$.

5.4 Analisi delle Distribuzioni Stazionarie

Come ultimo aspetto, vengono analizzate le distribuzioni stazionarie di probabilità degli stati, o *steady state probability distribution*, le quali forniscono un'indicazione di quale sia la probabilità di trovarsi nei diversi stati del sistema seguendo una certa politica.

Come si nota in figura 5.6 e 5.7, tutte le politiche mantengono la batteria del sensore su valori intermedi. Questo comportamento appare estremamente sensato. Mantenere la batteria su livelli di carica elevati potrebbe infatti rivelarsi un approccio eccessivamente conservativo, facendo diminuire le performance del sistema anche se non necessario. D'altro canto, è opportuno evitare di avvicinarsi a livelli energetici inferiori, in modo di scongiurare il pericolo di scaricare la batteria.

Parallelamente, aumentando la soglia di rischio, il sensore opererà con livelli di batteria medi più bassi, sfruttando la maggior tolleranza per massimizzare il

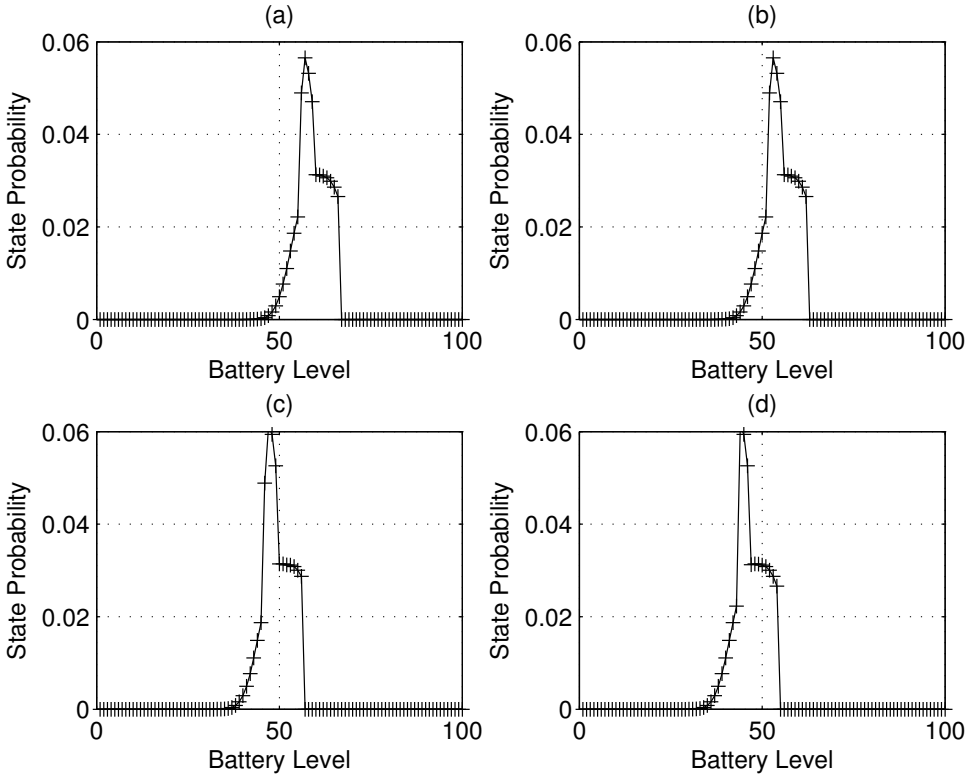

Figura 5.6: Steady state probability distribution per $x = 0$ e : (a) $\alpha = 0.1$, (b) $\alpha = 0.2$, (c) $\alpha = 0.3$, (d) $\alpha = 0.4$.

return.

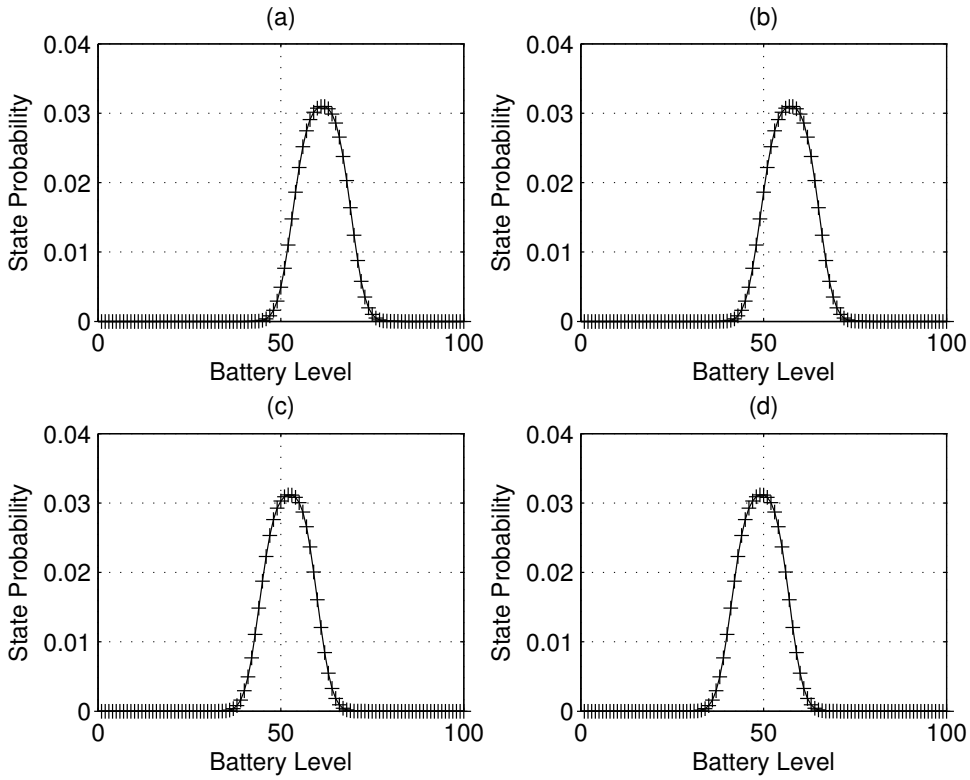

Figura 5.7: Steady state probability distribution per $x = 1$ e : (a) $\alpha = 0.1$, (b) $\alpha = 0.2$, (c) $\alpha = 0.3$, (d) $\alpha = 0.4$.

Capitolo 6

Conclusioni

In questo lavoro di tesi è stato sviluppato un algoritmo di apprendimento per rinforzo risk-sensitive. In seguito è stato applicato ad un modello di un sensore wireless alimentato da una fonte di energia rinnovabile. I risultati ottenuti mantengono le aspettative iniziali, e dimostrano la bontà delle scelte implementative adottate.

Ad ogni modo, come qualsiasi cosa, presenta dei margini di miglioramento. Per quanto riguarda l'algoritmo in sé stesso, il valore del parametro lagrangiano che determina l'influenza del rischio sulla politica ottima potrebbe essere aggiornato in modo differente, cosicché il suo valore ottimale venga raggiunto più velocemente.

Per quanto riguarda il modello del sistema invece, è possibile lavorare su più punti. Primo, può essere adottato un modello della fonte di energia solare a più stati, in grado di distinguere le varie fasi di irradiazione di luce che occorrono nell'arco della giornata, e magari introducendo una nozione relativa alle condizioni meteorologiche di un dato periodo dell'anno. Secondo, è auspicabile ottenere un modello più generale e completo del sensore, che comprenda la quasi totalità delle funzionalità del dispositivo e che tenga in considerazione la cooperazione e l'influenza reciproca tra nodi appartenenti alla stessa rete di sensori wireless.

Bibliografia

[Coraluppi and Marcus, 1999] Coraluppi, S., and Marcus, S.. Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. In *Automatica, 35, vol. 2*, pages 301–309. 1999.

[Filar et al., 1989] Filar, J.A, Kallenberg, L.C.M. and Lee, H. Variance-penalized Markov decision processes. In *Mathematics of Operations Research 14, vol. 1*, pages 147-161. INFORMS, 1989.

[Fulkerson et al., 1998] Fulkerson, M. S., Littman, M. L., and Keim, G. A.. Speeding safely: Multi-criteria optimization in probabilistic planning. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 831. MIT Press, 1998.

[Geibel, 2001] Geibel, P.. Reinforcement Learning with Bounded Risk. In *Proceedings of the 18th International Conference on Machine Learning*, pages 162-169. Morgan Kaufmann, 2001.

[Hans et al., 2008] Hans, A., Schneegaß, D., Schäfer, A. M., and Udluft, S.. Safe exploration for reinforcement learning. In *ESANN*, pages 143-148. 2008.

[Heger, 1994a] Heger, M..Consideration of Risk in Reinforcement Learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 105-111. Morgan Kaufmann, 1994.

[Heger, 1994b] Heger, M.. Risk and Reinforcement Learning: Concepts and Dynamic Programming. Technical Report, Universität Bremen, 1994.

[Howard and Matheson, 1972] Howard, R.A., and Matheson, J.E.. Risk-sensitive Markov decision processes. In *Management Science 18*, pages 356-369. INFORMS, 1972.

[Koenig and Simmons, 1994] Koenig, S., and Simmons, R.G.. Risk-sensitive planning with probabilistic decision graphs. In *KR'94: Principles of Knowledge Representation and Reasoning*, pages 363–373. Morgan Kaufmann, 1994.

[Polo and Rebollo, 2011] Polo, F.J.G., and Rebollo, F.F.. Safe Reinforcement Learning in High-Risk Tasks through Policy Improvement. In *Adaptive Dynamic Programming And Reinforcement Learning, 2011 IEEE Symposium on*, pages 76-83. IEEE, 2011.

[Rummery and Niranjan, 1994] Rummery, G. A., and Niranjan, M.. On-Line Q-Learning Using Connectionist Systems.Technical Report. 1994.

[Sutton and Barto, 1998] Sutton, R.S., and Barto, A.G.. Reinforcement Learning - An Introduction. MIT Press. 1998.

[Watkins, 1992] Watkins, C. J. C. H., and Dayan, P.. Q-learning. In *Machine Learning, 8, Special Issue on Reinforcement Learning*, pages 279-292. 1992.

[Zordan et al., 2012] Zordan, D., Martinez, B., Vilajosana, I., and Rossi, M.. On the Performance of Lossy Compression Schemes for Energy Constrained Sensor Networking. Submitted. 2012.