Intelligenza Artificiale e Machine Learning

Esercizi Svolti di Intelligenza Artificiale e Machine Learning

Anno Accademico: 2024/25

Giacomo Sturm

Dipartimento di Ingegneria Civile, Informatica e delle Tecnologie Aeronautiche
Università degli Studi "Roma Tre"

Sorgente del file LaTeX disponibile al seguente link:
https://github.com/00Darxk/Intelligenza-Artificiale-e-Machine-Learning

Indice

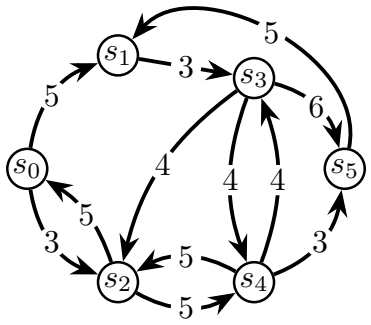
1 Esercizi di Intelligenza Artificiale

1.1 Esercizio 1

Dato il seguente Spazio degli Stati: $s_0, s_1, s_2, s_3, s_4, s_5$, con stato iniziale s_0, e stato obiettivo s_5. Questi stati sono collegati da certi operatori di costo:

- F: costo 3. $F(s_0) = s_2, F(s_1) = s_3, F(s_4) = s_5$;
- G: costo 4. $G(s_3) = s_2, G(s_3) = s_4, G(s_4) = s_3$;
- H: costo 5; $H(s_0) = s_1, H(s_2) = s_0, H(s_2) = s_4, H(s_4) = s_2, H(s_5) = s_1$;
- I: costo 6; $I(s_3) = s_5$.

Nel modo seguente:



E la seguente funzione euristica: $h(s_0) = 10$, $h(s_1) = 6$, $h(s_2) = 7$, $h(s_3) = 4$, $h(s_4) = 5$, $h(s_5) = 0$.

1. Eseguire l'algoritmo A* con modalità tree-search, fino alla terminazione disegnando l'albero di ricerca, riportando per ogni passo il contenuto della frontiera, il nodo scelto per l'espansione ed i nodi generati;
2. Riportare la soluzione trovata dell'algoritmo, indicando se tale soluzione è o non è quella ottima e indicando se la funzione euristica h rispetta la condizione di ammissibilità (motivare la risposta).

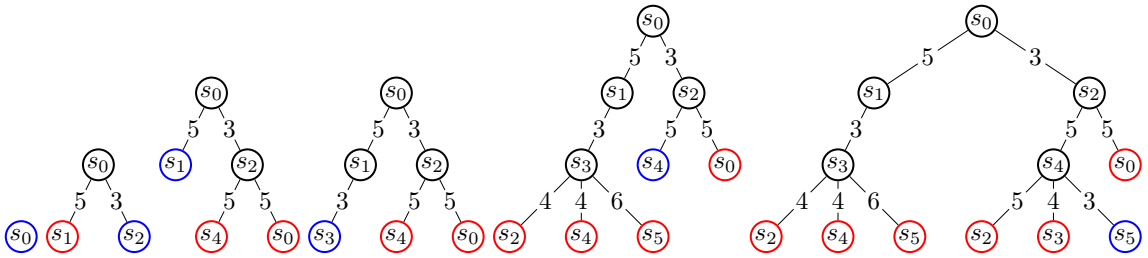
Domanda 1

Bisogna specificare i vari passi ottenuti dall'algoritmo A*, specificando la frontiera i nodi contenuti ed espansi ed il nodo successivo da espandere.

1. Nella frontiera è presente solamente lo stato iniziale $[s_0]$;

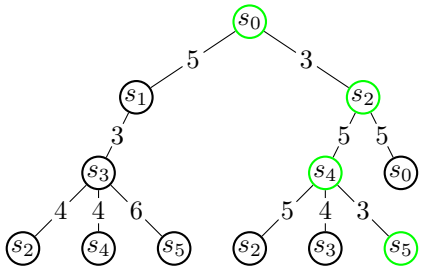
2. Si espande lo stato iniziale s_0, nella frontiera si ha $[s_2, s_1]$, dove $f(s_2) = 3+7$ e $f(s_1) = 5+6$;
3. Si espande il nodo s_2, con $f(s_2) = 10$, per cui nella frontiera entra il nodo s_4 e s_0: $[s_1, s_4, s_0]$, con $f(s_1) = 5+6$, $f(s_4) = 8+5$ e $f(s_0) = 8+10$;
4. Si espande il nodo s_1, con $f(s_1) = 12$, per cui nella frontiera entra il nodo s_3: $[s_3, s_4, s_0]$, con $f(s_3) = 8+4$, $f(s_4) = 8+5$ e $f(s_0) = 8+10$;
5. Si espande il nodo s_3, con $f(s_3) = 12$, per cui nella frontiera entra il nodo s_2, s_4 ed s_5: $[s_4(13), s_5, s_4(17), s_0, s_2]$, con $f(s_4) = 8+5$, $f(s_5) = 14+0$, $f(s_4) = 12+5$, $f(s_0) = 8+10$ ed $f(s_2) = 12+7$;
6. Si espande il nodo s_4 con $f(s_4) = 13$, per cui nella frontiera entra il nodo s_2, s_3 ed s_5: $[s_5(11), s_5(14), s_3, s_4, s_0, s_2(19), s_2(20)]$, con $f(s_5) = 11+0$, $f(s_5) = 14+0$, $f(s_3) = 12+4$, $f(s_4) = 12+5$, $f(s_0) = 8+10$, $f(s_2) = 12+7$, $f(s_2) = 13+7$;
7. Si espande il nodo s_5, con $f(s_5) = 11$. Questo nodo è il nodo obiettivo, quindi l'algoritmo termina.

I nodi rossi sono quelli nella frontiera, i nodi blu sono i nodi nella frontiera da espandere:



Domanda 2

La soluzione dell'algoritmo è s_0, s_2, s_4, s_5, di costo 11:



La funzione euristica h utilizzata è maggiore del costo effettivo per il nodo s_4:

$$h(s_4) = 5 > 3 = h^*(s_4) \tag{1.1}$$

Quindi non è una funzione ammissibile. La soluzione trovata non è garantito sia la soluzione ottima.

1.2 Esercizio 2

Dati i seguenti stati $s_0, s_1, s_2, s_3, s_4, s_5$, si hanno le seguenti operazioni:

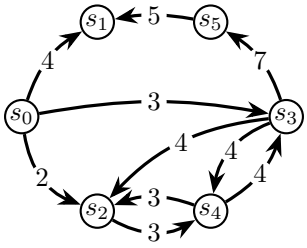
- F di costo 2: $F(s_0) = s_2$;
- G di costo 3: $G(s_0) = s_3$, $G(s_2) = s_4$, $G(s_4) = s_2$;
- H di costo 4: $H(s_0) = s_1$, $H(s_3) = s_2$, $H(s_3) = s_4$, $H(s_4) = s_3$;
- I di costo 5: $I(s_5) = s_1$;
- L di costo 7: $L(s_3) = s_5$;

Con la seguente funzione euristica: $h(s_0) = 10$, $h(s_1) = 7$, $h(s_2) = 7$, $h(s_3) = 5$, $h(s_4) = 4$ e $h(s_5) = 0$.

1. Disegnare lo spazio degli stati;
2. Eseguire l'algoritmo A* con modalità graph-search, fino alla terminazione disegnando l'albero di ricerca, riportando per ogni passo il contenuto della frontiera e della close, il nodo scelto per l'espansione ed i nodi generati;
3. Riportare la soluzione trovata dell'algoritmo, indicando se tale soluzione è o non è quella ottima e indicando se la funzione euristica h rispetta la condizione di ammissibilità (motivare la risposta).

Domanda 1

Si ha lo spazio degli stati:

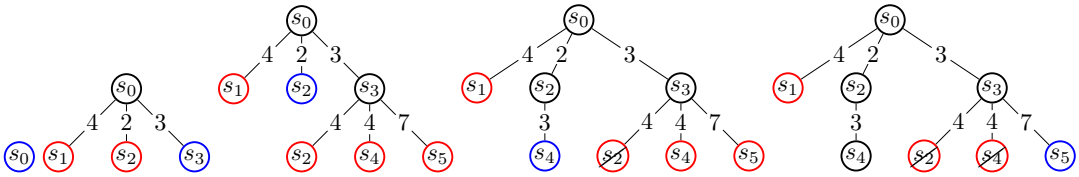


Domanda 2

Si effettua l'algoritmo A* in modalità graph-search, quindi quando viene espanso un nodo, il suo stato corrispondente viene inserito nella lista close e quando viene scelto per l'espansione viene scartato e si passa al passo successivo. I nodi relativi ai nodi già visitati dato che non verranno sicuramente espansi non vengono inseriti nella frontiera quando vengono generati da uno stato genitore. Si descrive il processo dell'algoritmo:

1. La close è vuota, nella frontiera è presente solo il nodo corrispondente allo stato iniziale $[s_0]$, e la close è vuota $[]$;
2. Viene estratto dalla frontiera ed espanso lo stato s_0 ed i suoi figli s_1, s_2 ed s_3 vengono inseriti nella frontiera $[s_3, s_2, s_1]$ con $f(s_3) = 8$, $f(s_2) = 9$ e $f(s_1) = 11$. La close contiene lo stato iniziale $[s_0]$;
3. Viene espanso il nodo s_3 e vengono inseriti nella frontiera i suoi figli s_2, s_4 e s_5. Bisogna distinguere i due nodi associati entrambi allo stato s_2, tramite il valore della loro funzione di valutazione: $[s_2(9), s_5, s_1, s_4, s_2(14)]$ con $f(s_2) = 8$, $f(s_5) = 10$, $f(s_1) = 11$, $f(s_4) = 11$ e $f(s_2) = 14$. Mentre la close contiene i nodi $[s_0, s_3]$;
4. Viene espanso il nodo s_2 e si inserisce il suo nodo figlio nella frontiera, specificando il valore per distinguerlo dall'altro nodo associato allo stato s_4. Poiché lo stato s_2 è già stato visitato si potrebbe rimuovere dalla frontiera, poiché anche se venisse scelto non verrebbe espanso. La frontiera contiene i nodi $[s_4(9), s_5, s_1, s_4(11), \cancel{s_2}]$, con $f(s_4) = 9$, $f(s_5) = 10$, $f(s_1) = 11$, $f(s_4) = 11$ e $f(s_2) = 14$. La close contiene i nodi $[s_0, s_3, s_2]$;
5. Viene espanso il nodo s_4, i suoi nodi corrispondono a stati presenti nella close, quindi non vengono inseriti nella frontiera. La frontiera è quindi $[s_5, s_1, \cancel{s_4}, \cancel{s_2}]$. La close contiene i nodi $[s_0, s_3, s_2, s_4]$;
6. Viene estratto il nodo s_5, questo rappresenta lo stato obiettivo, quindi l'algoritmo termina.

In nero sono indicati i nodi espansi, quindi all'interno della close, in rosso i nodi all'interno della frontiera ed in blu i nodi scelti per l'espansione nel passo corrente, l'algoritmo termina quando viene scelto il nodo associato allo stato s_5 per l'espansione:



Domanda 3

La soluzione individuata è $s_0 \rightarrow s_3 \rightarrow s_5$, di costo 10. Queste rappresenta la soluzione ottima. La funzione euristica è ammissibile, poiché per ogni stato s_i la funzione euristica $h(s_i)$ è minore, o uguale per gli stati s_0 e s_5, al costo effettivo del cammino dal nodo s_i al nodo obiettivo s_5.

1.3 Esercizio 3

Scrivere il codice Python per la gestione di una lista concatenata ordinata, avvalendosi delle classi.

```
class Node:
    data = None
    next = None

    def __init__(self, data, next):
        self.data = data
        self.next = next

class List:
    def __init__(self):
        self.__head = None
        self.__tail = None

    def add(self, newNode):
        if self.__head == None or self.__head.data > newNode.data:
            if self.__head == None:
                self.__tail = newNode
            newNode.next = self.__head
            self.__head = newNode

        elif self.__tail.data < newNode.data:
            if self.__tail != None:
                self.__tail.next = newNode
            if self.__head == None:
                self.__head = NewNode
            self.__tail = newNode

        else:
            p = self.__head
            while p.next != None and p.next.data < newNode.data:
                p = p.next
            newNode.next = p.next
            p.next = newNode
```

1.4 Esercizio 4

Illustrare nel dettaglio l'algoritmo "Greedy", presentando il codice Python, opportunamente commentato per il problema della ricerca di un itinerario.

Per realizzare l'algoritmo sono necessarie le classi per gli stati, individuati solamente dal loro nome nello spazio degli stati, ed i nodi dell'albero di ricerca:

```
# nodo dell'albero di ricerca: contenente lo stato, il nodo genitore
# ed il valore della funzione euristica per lo stato
class Node:
    def __init__(self, state, parent, h):
        self.state = state
        self.parent = parent
        self.h = h

    # funzione per stampare la soluzione
    def printPath(self):
        if self.parent != None:
            self.parent.printPath()
        print("->", self.state.name)

# stato del problema
class State:
    # crea uno stato, definito solamente dal nome
    # se non viene specificato crea lo stato iniziale
    def __init__(self, name = None):
        if name == None:
            self.name = self.getInitialState()
        else:
            self.name = name

    # restituisce lo stato iniziale definito a priori
    def getInitialState(self):
        initialState = statoIniziale
        return initialState

    # ottiene i successori dal dizionario 'connections'
    def successorFunction(self):
        return connections[self.name]

    # verifica se è uno stato obiettivo definito a priori
    def checkGoalState(self):
        return self.name == statoObiettivo
```

La funzione euristica ed i nodi successori per un determinato stato si possono implementare allo stesso modo tramite un dizionario, dove la chiave è il nome dello stato, ed il valore è o il valore dell'euristica o la lista degli stati successori:

```
# dizionario dei successori:
connections['A'] = ['B', 'C']
connections['B'] = ['C']
connections['C'] = ['B', 'D']
connections['D'] = ['B', 'C', 'E']
connections['E'] = ['A', 'C']
# dizionario delle euristiche
h['A'] = 10
h['B'] = 9
h['C'] = 6
h['D'] = 3
h['E'] = 0
```

L'algoritmo inizializza la frontiera con coda di priorità, contenente solamente l'elemento corrispondente alla radice. In seguito continua ad iterare fino a quando non svuota la frontiera, rimuovendo il primo elemento in frontiera, rappresentati come tuple euristica e nodo. In questo modo si possono ordinare in base al loro valore di euristica. Se il nodo estratto è un obiettivo, termina l'algoritmo e stampa la soluzione, altrimenti viene espanso ed i suoi figli vengono inseriti all'interno della frontiera.

```
import queue

def Greedy_Best_First():
    # crea la frontiera con una priority queue
    fringe = queue.PriorityQueue()

    # crea lo stato iniziale
    initialState = State()

    # crea la radice dell'albero di ricerca e la aggiunge alla frontiera
    root = Node(initialState, None, h[initialState.name])
    fringe.put((root.h, root))

    while not fringe.empty():
        # si itera prendendo il primo elemento della frontiera
        # fino all'esaurimento dei suoi elementi
        (currentH, currentNode) = fringe.get()

        # si controlla se corrisponde ad uno stato obiettivo
        if currentNode.state.checkGoalState():
            currentNode.state.printPath()
            break
        # altrimenti si espande e si aggiungono i suoi figli alla frontiera
        else:
```

```
childStates = currentNode.state.successorFunction()
# crea gli elementi corrispondenti per l'aggiunta in frontiera
for childState in childStates:
    childNode = Node(State(childState), currentNode,
                         h[State(childState).name])
    fringe.put((childNode.h, childNode))
```

2 Esercizi di Machine Learning: Classificatore di Bayes

2.1 Esercizio 1

Stimare la classe di appartenenza del pattern $\mathbf{x} = (57, 168)$, nel seguente dataset, utilizzando il classificatore di Bayes:

Peso	Altezza	Classe
72	173	w_1
54	159	w_1
65	172	w_1
58	170	w_1
62	165	w_1
72	176	w_1
60	173	w_1
64	179	w_1

Peso	Altezza	Classe
55	166	w_2
46	158	w_2
54	158	w_2
47	160	w_2
55	167	w_2
54	166	w_2
55	164	w_2
49	157	w_2
51	165	w_2
51	162	w_2

Si determina il vettore medio delle due classi:

$$\boldsymbol{\mu}_1 = \frac{1}{8}\sum_{i=1}^{8} \mathbf{x}_i \in w_1 = \begin{bmatrix} 63.4 \\ 170.9 \end{bmatrix}$$

$$\boldsymbol{\mu}_2 = \frac{1}{10}\sum_{i=1}^{10} \mathbf{x}_i \in w_2 = \begin{bmatrix} 51.5 \\ 162.3 \end{bmatrix}$$

E la loro matrici di covarianza:

$$\Sigma_1 = \frac{1}{8}\sum_{i=0}^{8} (\mathbf{x}_i \in w_1 - \boldsymbol{\mu}) \cdot (\mathbf{x}_i \in w_1 - \boldsymbol{\mu})^T = \begin{bmatrix} 35.2 & 23.3 \\ 23.3 & 34.9 \end{bmatrix}$$

$$\Sigma_2 = \frac{1}{10}\sum_{i=0}^{10} (\mathbf{x}_i \in w_2 - \boldsymbol{\mu}) \cdot (\mathbf{x}_i \in w_2 - \boldsymbol{\mu})^T = \begin{bmatrix} 10.1 & 8.9 \\ 8.9 & 13.0 \end{bmatrix}$$

Si possono stimare le probabilità a priori come:

$$P(w_1) = \frac{\dim \Omega_1}{\dim \Omega} = \frac{8}{18} = 0.44$$

$$P(w_1) = \frac{\dim \Omega_2}{\dim \Omega} = \frac{10}{18} = 0.56$$

In caso non sia possibile determinare le probabilità a priori, si possono considerare le classi equiprobabili. La probabilità condizionale si ottiene dalla gaussiana inserendo i parametri per le due classi:

$$p(\mathbf{x}|w_1) = \frac{1}{2\pi|\Sigma_1|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma_1 (\mathbf{x} - \boldsymbol{\mu}_1)\right] = 0.0033$$

$$p(\mathbf{x}|w_2) = \frac{1}{2\pi|\Sigma_2|^{1/2}}\exp\left[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T\Sigma_2(\mathbf{x}-\boldsymbol{\mu}_2)\right] = 0.0045$$

La media pesata di queste rispetto alla probabilità delle classi rappresenta la probabilità assoluta stimata:

$$p(\mathbf{x}) = p(\mathbf{x}|w_1)\cdot P(w_1) + p(\mathbf{x}|w_2)\cdot P(w_2) = 0.0033\cdot 0.44 + 0.0045\cdot 0.56 = 0.004$$

Si ottengono poi le due probabilità a posteriori con il teorema di Bayes, dopo aver controllato che la probabilità di ognuna delle due classi sommata sia pari ad uno:

$$\begin{gathered} P(w_1|\mathbf{x}) = \frac{p(\mathbf{x}|w_1)\cdot P(w_1)}{p(\mathbf{x})} = \frac{0.0033\cdot 0.44}{0.004} = 0.36 \\ P(w_2|\mathbf{x}) = \frac{p(\mathbf{x}|w_2)\cdot P(w_2)}{p(\mathbf{x})} = \frac{0.0045\cdot 0.56}{0.004} = 0.64 \\ P(w_1|\mathbf{x}) + P(w_2|\mathbf{x}) = 1 \\ P(w_2|\mathbf{x}) > P(w_1|\mathbf{x}) \implies \mathbf{x}\in w_2 \end{gathered} \tag{2.1}$$

Il pattern $\mathbf{x}$ quindi appartiene alla classe w_2.