Andrea Augello
Department of Engineering, University of Palermo, Italy

Ricerche informate ed euristiche

Università degli Studi di Palermo

Ricerche informate ed euristiche

La scorsa esercitazione abbiamo visto le ricerche in profondità e a costo uniforme, e le abbiamo applicate a problemi di soddisfacimento di vincoli.

In questa esercitazione vedremo le ricerche informate, che utilizzano euristiche per guidare la ricerca verso le soluzioni.

Problema di esempio

Ricerca in ampiezza

Ricerca in profondità

Ricerca ad iterazione della profondità

Ricerca greedy

Algoritmo A*

Problema di esempio

Problema di esempio

Come problema di esempio consideriamo il puzzle dell'otto. L'obiettivo è quello di spostare i numeri da 1 a 8 in modo da ottenere la configurazione finale:

2	4	3
7	1	5
8		6

1	2	3
4	5	6
7	8	

Problema di esempio

Per rappresentare lo stato utilizziamo una lista, rappresentando lo spazio vuoto con il numero 0:

```
start=[2,4,3,7,1,5,8,0,6]
goal =[1,2,3,4,5,6,7,8,0]
```

Agente

Rispetto alle esercitazioni precedenti, l'agente è stato modificato per tenere traccia del numero di stati generati per confrontare le prestazioni degli algoritmi.

```
class Agent():
def __init__(self, start, goal=[1, 2, 3, 4, 5, 6, 7, 8, 0]):
    self.generated_states = 0
    self.start = start
    self.goal = goal
    # Lo spazio vuoto può essere spostato in 4 direzioni
    self.moves = [[1, 0], [-1, 0], [0, 1], [0, -1]]
    self.frontier = [[start]]

def next_paths(self, path):
    pass

def move(self, state, move):
    pass
```

Generazione degli stati successivi

- `next_paths` è una funzione che prende in input un percorso e restituisce una lista di percorsi ottenuti aggiungendo uno stato successivo allo stato finale del percorso.
- `move` è una funzione che prende in input uno stato e una mossa e restituisce lo stato ottenuto applicando la mossa allo stato, se la mossa è applicabile, altrimenti restituisce `None`.
  - Dopo la mossa lo spazio vuoto non può trovarsi in una posizione fuori dalla griglia.
  - Se lo spazio vuoto si trova in una posizione adiacente al bordo, non è possibile spostarlo verso il bordo.

Generazione degli stati successivi

```
def next_paths(self, path):
    state = path[-1]
    for move in self.moves:
        new_state = self.move(state, move)
        if new_state and new_state not in path:
            yield path + [new_state]
```

Generazione degli stati successivi

```
def next_paths(self, path):
    state = path[-1]
    for move in self.moves:
        new_state = self.move(state, move)
        if new_state and new_state not in path:
            yield path + [new_state]
```

```
def move(self, state, move):
    empty = state.index(0)
    new_empty = empty + move[0] + 3 * move[1]
```

Generazione degli stati successivi

```
def next_paths(self, path):
    state = path[-1]
    for move in self.moves:
        new_state = self.move(state, move)
        if new_state and new_state not in path:
            yield path + [new_state]
```

```
def move(self, state, move):
    empty = state.index(0)
    new_empty = empty + move[0] + 3 * move[1]
    if (new_empty < 0 or new_empty > 8) or \
            (empty % 3 == 0 and move[0] == -1) or \
            (empty % 3 == 2 and move[0] == 1):
        return None
    new_state = state.copy()
```

Generazione degli stati successivi

```
def next_paths(self, path):
    state = path[-1]
    for move in self.moves:
        new_state = self.move(state, move)
        if new_state and new_state not in path:
            yield path + [new_state]
```

```
def move(self, state, move):
    empty = state.index(0)
    new_empty = empty + move[0] + 3 * move[1]
    if (new_empty < 0 or new_empty > 8) or \
            (empty % 3 == 0 and move[0] == -1) or \
            (empty % 3 == 2 and move[0] == 1):
        return None
    new_state = state.copy()
    new_state[empty], new_state[new_empty] = new_state[new_empty],
       new_state[empty]
    return new_state
```

Funzione di utilità

```
def state_to_str(state):
    return "\n".join([" ".join([str(state[3 * i + j]) for j in
        range(3)]) for i in range(3)])
```

Prende in input una lista monodimensionale di 9 elementi e restituisce una stringa che rappresenta la griglia 3x3.

Una volta ottenuto un percorso, possiamo stampare i vari stati con:

```
print(*[state_to_str(state) for state in path], sep="\n\n")
```

Ricerca in ampiezza

Ricerca in ampiezza

La ricerca in ampiezza è già stata implementata nell'esercitazione 7.

L'unica differenza è che adesso, prima di aggiungere i successori di un nodo alla frontiera, vengono contati ed sommati a `self.generated_states`.

Ricerca in ampiezza

La ricerca in ampiezza è già stata implementata nell'esercitazione 7.

L'unica differenza è che adesso, prima di aggiungere i successori di un nodo alla frontiera, vengono contati ed sommati a `self.generated_states`.

Performance:

- Tempo: 0.1648 s (processore Intel Core i5-104000 a 2.90 GHz)
- Spazio massimo utilizzato: 362 KB
- Nodi calcolati: 720

Ricerca in profondità

Ricerca in profondità

La ricerca in profondità è già stata implementata nell'esercitazione 8. La frontiera conterrà sempre un solo percorso, che verrà esteso ad ogni iterazione. Il parametro `depth` indica la profondità massima della ricerca.

```
    def dfs(self, depth=0):
        if depth:
            path = self.frontier.pop(0)
            if path[-1] == self.goal:
                yield path
            for next_path in self.next_paths(path):
                self.frontier = [next_path]
                self.generated_states += 1
                yield from self.dfs(depth=depth - 1)
```

Performance:

- L'interprete Python non ottimizza la ricorsione, la ricerca in profondità va in crash se non viene specificata una profondità massima.

Ricerca ad iterazione della profondità

Ricerca ad iterazione della profondità

In un ciclo chiamiamo la ricerca in profondità con una profondità massima crescente.

Ricerca ad iterazione della profondità

In un ciclo chiamiamo la ricerca in profondità con una profondità massima crescente.

Performance:

- Tempo: 0.3082 s
- Spazio massimo utilizzato: 4 KB
- Nodi calcolati: 1696

Ricerca greedy

Ricerca greedy

Iniziamo a vedere ricerca informate che fanno uso di euristiche. Useremo una euristica molto semplice: il numero di caselle fuori posto.

Ricerca greedy

Iniziamo a vedere ricerca informate che fanno uso di euristiche. Useremo una euristica molto semplice: il numero di caselle fuori posto.

```
    def heuristic(self, state):
        # Calcola il numero di tessere fuori posto
        return sum([1 if state[i] != self.goal[i] else 0 for i in
            range(len(state))])
```

Ricerca greedy

Per implementare la ricerca greedy, modifichiamo l'algoritmo di ricerca in ampiezza in modo che la frontiera sia ordinata in base al valore dell'euristica prima di effettuare ogni chiamata ricorsiva.

Ricerca greedy

Per implementare la ricerca greedy, modifichiamo l'algoritmo di ricerca in ampiezza in modo che la frontiera sia ordinata in base al valore dell'euristica prima di effettuare ogni chiamata ricorsiva.

Performance:

- Tempo: 0.0160 s
- Spazio massimo utilizzato: 6 KB
- Nodi calcolati: 26

Algoritmo A*

Algoritmo A*

La ricerca A* aggiunge all'euristica anche il costo del percorso.

Algoritmo A*

La ricerca A* aggiunge all'euristica anche il costo del percorso.

Performance:

- Tempo: 0.0400 s
- Spazio massimo utilizzato: 12 KB
- Nodi calcolati: 44