Andrea Augello
Department of Engineering, University of Palermo, Italy

Ricerche informate ed euristiche

Università degli Studi di Palermo

Ricerche informate ed euristiche

In questa esercitazione vedremo ulteriori algoritmi di ricerca non informata e 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	
8	6	5

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

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)]).replace("0", " ")
```

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:

```
for state in solution:
    print(state_to_str(state), end="\r\033[2A", flush=True)
    sleep(0.5)
```

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
    self.moves = [[1, 0], [-1, 0], [0, 1], [0, -1]]

def move(self, state, move):
    return state

def next_paths(self, path):
    self.generated_states += 1
    yield path

def search(self):
    raise NotImplementedError
```

Generazione degli stati successivi

- `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.
- `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.

Generazione degli stati successivi

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

Generazione degli stati successivi

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

```
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):
    for move in self.moves:
        if (new := self.move(path[-1], move)) and new not in path:
            self.generated_states += 1
            yield path + [new]
```

```
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):
    for move in self.moves:
        if (new := self.move(path[-1], move)) and new not in path:
            self.generated_states += 1
            yield path + [new]
```

```
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
```

Ricerca in ampiezza

Ricerca in ampiezza

La ricerca in ampiezza è già stata discussa in esercitazioni precedenti.

Ricerca in ampiezza

La ricerca in ampiezza è già stata discussa in esercitazioni precedenti.

Performance:

- Tempo: 0.0226 s
- Spazio massimo utilizzato: 446 KB
- Nodi calcolati: 2364
- Profondità della soluzione: 12

Ricerca in profondità

Ricerca in profondità

La ricerca in profondità l'abbiamo già vista in precedenza. Come unica differenza, stavolta la funzione accetta un parametro `depth` che rappresenta la profondità massima della ricerca, in modo da evitare di andare troppo in profondità e non riuscire a trovare la soluzione.

Ricerca in profondità

La ricerca in profondità l'abbiamo già vista in precedenza. Come unica differenza, stavolta la funzione accetta un parametro `depth` che rappresenta la profondità massima della ricerca, in modo da evitare di andare troppo in profondità e non riuscire a trovare la soluzione.

Performance:

- Tempo: 0.4962 s
- Spazio massimo utilizzato: 14 KB
- Nodi calcolati: 43607
- Profondità della soluzione: 20

Ricerca ad iterazione della profondità

Ricerca ad iterazione della profondità

Estendiamo `AgentDFS` per implementare la ricerca ad iterazione della profondità. In un while infinito, reimpostiamo lo stato iniziale, incrementiamo la profondità ad ogni iterazione, e chiamiamo la ricerca della classe genitore con la profondità corrente.

Ricerca ad iterazione della profondità

Estendiamo `AgentDFS` per implementare la ricerca ad iterazione della profondità. In un while infinito, reimpostiamo lo stato iniziale, incrementiamo la profondità ad ogni iterazione, e chiamiamo la ricerca della classe genitore con la profondità corrente.

Performance:

- Tempo: 0.0527 s
- Spazio massimo utilizzato: 8 KB
- Nodi calcolati: 5630
- Profondità della soluzione: 12

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.

Implementiamo una funzione `heuristic` che prende in input uno stato e restituisce il valore dell'euristica.

Ricerca greedy

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

Implementiamo una funzione `heuristic` che prende in input uno stato e restituisce il valore dell'euristica.

Indizio: per ogni casella, se stato e goal non coincidono, incrementare un contatore (si può usare la funzione `zip`, o fare tutto con una list comprehension e `sum`).

Ricerca greedy

Per implementare la ricerca greedy, estendiamo AgentBFS modificando la ricerca ordinando la frontiera in base al valore dell'euristica prima di effettuare ogni chiamata ricorsiva.

Ricerca greedy

Per implementare la ricerca greedy, estendiamo AgentBFS modificando la ricerca ordinando la frontiera in base al valore dell'euristica prima di effettuare ogni chiamata ricorsiva.

Performance:

- Tempo: 0.0011 s
- Spazio massimo utilizzato: 4 KB
- Nodi calcolati: 31
- Profondità della soluzione: 12

Nota: la ricerca greedy non garantisce di trovare la soluzione ottima.

Algoritmo A*

Algoritmo A*

La ricerca A* aggiunge all'euristica anche il costo del percorso. Basta estendere la classe AgentGreedy e modificare il metodo `heuristic`.

Algoritmo A*

La ricerca A* aggiunge all'euristica anche il costo del percorso. Basta estendere la classe AgentGreedy e modificare il metodo `heuristic`.

Performance:

- Tempo: 0.0800 s
- Spazio massimo utilizzato: 17 KB
- Nodi calcolati: 111
- Profondità della soluzione: 12

Confronti riassuntivi

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

Algoritmo	Tempo (s)	Spazio (KB)	Nodi	Profondità
BFS	0.0226	446	2364	12
DFS	0.4962	14	43607	20
IDS	0.0527	8	5630	12
Greedy	0.0011	4	31	12
A*	0.0800	17	111	12

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

Algoritmo	Tempo (s)	Spazio (KB)	Nodi	Profondità
BFS	0.0326	681	3486	13
DFS	0.0135	14	1211	17
IDS	0.0756	9	8208	13
Greedy	6.1580	933	3717	45
A*	0.0316	36	214	13