

Heuristic Estimation

Greedy Search

....

UCS Behavi

A* Behavior

A* Admissib

A* Optimality

Heuristic Search



Dijkstra

Heuristic Est

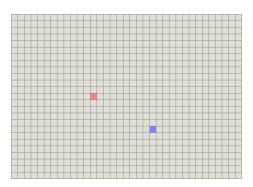
Greedy Sea

....

UCS Behav

A* Behavio

A* Optimality



Dijkstra's Algorithm Behavior

start state: red, goal state: blue



Dijkstra

Greedy Sea

Greedy Se

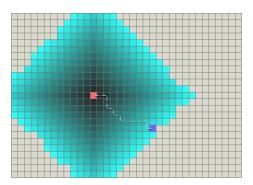
A* Search

UCS Behav

A* Behavio

A* Optimality

Dijkstra's Algorithm Behavior



generated nodes: cyan (darker = earlier in priority queue)



Heuristic Estimation

Greedy Search 8-Puzzle A* Search

UCS Behavior

A* Admissib A* Optimalit

Heuristic Estimation

Many of the generated nodes are unnecessary – they can't possibly be part of a good solution.

Heuristic knowledge is an estimation or educated guess, but not necessarily correct

Heuristic algorithms use heuristic knowledge to solve a problem

Heuristic functions (or just *heuristics*) take a state as an input and outputs an estimate of the cost to a goal state



Heuristic Estimatic

Greedy Search

0.0 1

A# Sanrah

UCS Beha

A # Rabavia

A R A desired

A* Optimality

1: *Open* ← ordered list containing initial state
2: while true do

z: wille due do

Greedy Search

3: **if** *Open* is empty **then**

4: **return** failure

5: $Node \leftarrow Open.Pop()$

6: **if** *Node* is goal **then**

7: **return** *Node* (or path to *Node*)

8: **else**

9: $Children \leftarrow Expand(Node)$

10: Add *Children* to *Open*, sorting on heuristic



Heuristic Estimatic

8-Puzzle

UCS Behav

A* Admissi

A* Optimality

8-Puzzle

h(n) = number of tiles out of place

Start state:
$$\begin{pmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & \sqcup & 5 \end{pmatrix}$$
 Goal state: $\begin{pmatrix} 1 & 2 & 3 \\ 8 & \sqcup & 2 \\ 7 & 6 & 5 \end{pmatrix}$

Draw a level of the search tree from expanding the start node, and determine h(n) for the child states.



Greedy Evaluation

Heuristic Search

8-Puzzle

Assume branching factor b and solution depth d

Completeness:

Time:

Space:

Admissibility:



Dijkstra Heuristic Estimatio

8-Puzzle

A* Search

- UCS Behavi
- A* Behavi
- A Admission
- A* Optimality
 Heuristic Function

A* Search

f(n) = g(n) + h(n): g(n) is cost from start state to current state, h(n) is heuristic estimate from current state to goal

- 1: $Open \leftarrow priority queue containing initial state$
- 2: while true do
- 3: **if** *Open* is empty **then**
- 4: **return** failure
- 5: $Node \leftarrow Open.Pop()$
- 6: **if** *Node* is goal **then**
- 7: **return** *Node* (or path to *Node*)
- 8: **else**
- 9: $Children \leftarrow Expand(Node)$
- 10: Add *Children* to *Open*, sorted on f(n)



Dijkstra Heuristic Estimatio

8-Puzzl

A* Search

.........

A* Optimality

.. ...

Assume branching factor b and solution depth d

Completeness:

Time:

Space:

Admissibility:

A* Evaluation



UCS Behavior

Heuristic Search

Dijkstra

Heuristic Estimation

8.Puzzle

A* Search

UCS Behavior

A* Behavior

A* Optimalit

Hamilaia Pos



- 10/14 -



A* Behavior

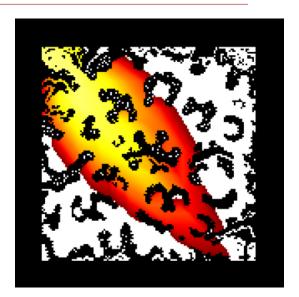
Heuristic Search

Dijkstra Heuristic Estimatio

A* Search

A* Behavior

A* Optimality



- 11/14 -



Dijkstra Heuristic Estimatio Greedy Search

A* Search

UCS Behav

A* Admissibility

A* Optimality

A* Admissibility

- \blacksquare Assume A* finds a goal node *s* using an admissible heuristic.
- Suppose node p is a node on a path to a better solution b.

$$\blacksquare f(p) = g(p) + h(p) \leq f(b) = g(b)$$

- But $g(s) = f(s) \le f(p)$
- Therefore $g(s) \le g(b)$ and s is optimal.



Dijkstra Heuristic Estimatio

8-Puzzle

UCS Behavi

A* Admissibili

A* Optimality

Heuristic Function

- \blacksquare For admissible h, nodes in open list can be sorted on f
- A* expands these nodes in sorted order
- A* must examine all nodes with $f < f^*$

A* Optimality



Dijkstra
Heuristic Estimation
Greedy Search
8-Puzzle

A* Search

UCS Behavior

A* Behavior

A* Admissibility

A* Optimality

Heuristic Function

Heuristic Functions

Function values are based on solving a simpler related problem. Function design:

- Relaxation: fewer or weaker constraints
- Abstraction: simplify search space by leaving out details

Admissibility:

- We want h(n) to give the largest value that does not exceed the actual cost to the goal of our original problem
- If h(n) never overestimates the cost to the goal, it is admissible and A* gives an optimal solution
- If $h_1(n) \le h_2(n) \forall n$, h_2 dominates h_1 and is a better heuristic function

We want h(n) to be easy to compute