Informed search

- **Use domain knowledge!**
  - Are we getting close to the goal?
  - Use a heuristic function that estimates how close a state is to the goal
  - A heuristic does NOT have to be perfect!
  - Example of strategies:
    1. Greedy best-first search
    2. A* search
    3. IDA*

启发函数不需要很完美

Greedy search

- Evaluation function $h(n)$ (heuristic)
- $h(n)$ estimates the cost from n to the goal
- Example: $h_{SLD}(n)$ = straight-line distance from n to Sault Ste Marie
- Greedy search expands the node that **appears** to be closest to goal

Greedy search: Pseudo-code

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

→没什么用，因为h(n)是固定的

A* search

- Minimize the total estimated solution cost
- Combines:
  - $g(n)$: cost to reach node n
  - $h(n)$: cost to get from n to the goal
  - $f(n) = g(n)+h(n)$

$f(n)$ is the estimated cost of the cheapest solution through n

A* search : Pseudo-code

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

与Greedy search相比，仅更改了cost函数.

Admissible heuristics

- An **admissible** heuristic never overestimates the cost to reach the goal, that is it is **optimistic**
- A heuristic h is admissible if

$$\forall node\ n,\ h(n) \leq h^*(n)$$

  where h^* is true cost to reach the goal from n.
- h_{SLD} (used as a heuristic in the map example) is admissible because it is by definition the shortest distance (straight line) between two points.

A* Optimality

If $h(n)$ is admissible, A* using tree search is optimal.

Rationale:

- Suppose G_o is the optimal goal.
  Suppose G_s is some suboptimal goal.
  Suppose n is on the shortest path to Go.
- $f(G_s) = g(G_s)$ since $h(G_s) = 0$
  $f(G_o) = g(G_o)$ since $h(G_o) = 0$
  $g(G_s) > g(G_o)$ since G_s is suboptimal
  Then $f(G_s) > f(G_o)$ . . . (1)
- $h(n) \leq h^*(n)$ since h is admissible
  $g(n)+h(n) \leq g(n)+h^*(n) = g(G_o) = f(G_o)$
  Then, $f(n) \leq f(G_o)$ . . . (2)

From (1) and (2) $f(G_s) > f(n)$, so A* will never select G_s during the search and hence A* is optimal.

We always will go toward Go rather than to go Gs.

G指的是到goal的序列

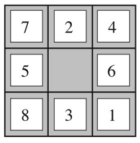
A* : PF Metrics

- **Complete:** Yes.
- **Time:** exponential
- **Space:** keeps every node in memory, the biggest problem
- **Optimal:** Yes!

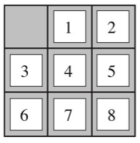
Search Efficiency of Heuristics

Heuristics for 8-puzzle

- The solution is 26 steps long.
- $h_1(n)$ = number of misplaced tiles
- $h_1(n)$ = 8
- $h_2(n)$ =total Manhattan distance (sum of the horizontal and vertical distances).
- Tiles 1 to 8 in the start state gives: h_2 = 3+1+2+2+2+3+3+2 = 18 which does not overestimate the true solution.



Start State

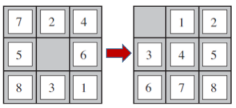


Goal State

- $h_{mis}(s)$ = #misplaced titles ∈ [0,8]: **Admissible**.
- $h_{1stp}(s)$ = #(1-step move) to reach the goal configuration: **Admissible**.

➢ $h_{1stp}(s) \geq h_{mis}(s) \Rightarrow h_{1stp}(s)$ is **'better'** than $h_{mis}(s)$.

What does 'better' mean?



Dominance

- For **admissible** h_1 and h_2, if $h_1(s) \geq h_2(s)$ for $\forall s$
  $\Rightarrow h_1$ **dominates** h_2 and is **more efficient** for search.
- **Theorem**: For any admissible heuristics h_1 and h_2, define
  $$h(s) = \max\{h_1(s), h_2(s)\}$$
  $h(s)$ is admissible and dominates both h_1 and h_2.

- **'Better' heuristic = dominance = better search efficiency.**
- **Question**: Which one to choose from a collection of admissible heuristics $h_1, \cdots, h_m$ & none dominates any other?

- **Answer**: $h(s) = \max\{h_1(s), \cdots, h_m(s)\}$ dominates all the others.

取大的能防止过多搜索，提高效率

Quantify Search Efficiency

- **Effective Branching Factor b^***: For a solution from A*, calculate b^* satisfying: $N = b^* + (b^*)^2 + \cdots + (b^*)^d$
  - N: #nodes of the solution,
  - d: depth of the solution tree.
  - E.g., A* finds a solution at depth 5 using 52 nodes ⇒ $b^* = 1.92$.
- Good heuristics have b^* close to 1 ⇒ large problems solved at reasonable computational cost.
- **b^* quantifies search efficiency of heuristics.**

希望 b^* 越小越好

Empirical: Factor b^*

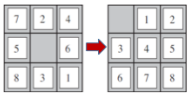
- **Aim**: Compare h_1 and h_2 regarding the search efficiency.
- **Setting**: Generate 1200 random problems with $d = \{2, \cdots, 24\}$ and solve them with IDS and A* with h_1 & h_2.
- **Note**: IDS – a baseline.

Generate Admissible Heuristics

① from Relaxed Problems

For 8-puzzle problem:

- **Real Rule**: A tile can only move to the **adjacent empty** square.
- **Relaxed rules**: h_{mis} and h_{1stp} are admissible
  - R1: A tile can move **anywhere** ⇒ $h_{mis}(s)$ = #(misplaced titles).
  - R2: A tile can move one step in **any direction** regardless of an occupied neighbour ⇒ $h_{1stp}(s)$ = #(1-step move) to reach goal.
- **Optimal solutions to problems with R1, R2 are easier to find.**



Relaxed Problem

- **Relaxed problem**: a problem with **relaxed rules** on the action.
- E.g. 8-puzzle problems with R1 and R2.

- **Theorem**: The cost of an optimal solution to **a relaxed problem** is an **admissible heuristic** for the original problem.
- No wonder h_{mis} and h_{1stp} are admissible.

② from sub-problems

- **Subproblem**
  - **Task**: get tiles 1, 2, 3 and 4 into their correct positions.
  - **Relaxation**: move them disregarding the others.
- **Theory**: cost*(subproblem)<cost*(original).
  - cost*(subproblem): the cost of the optimal solution of this subproblem.

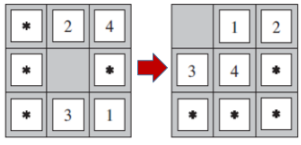


Fig.1. A subproblem of 8-puzzle.

Subproblem and Addmissible Heuristics

- **Admissible $h^*_{sub}(s)$**: estimate the cost from s to the subproblem goal.
  - E.g. $h^{(1,2,3,4)}_{sub}$ is the cost to solve the 1-2-3-4 subproblem.

- **Theorem**: $h_{sub}(s)$ dominates $h_{1stp}(s)$,
  - $h_{sub}(s) = max\{h^{(1,2,3,4)}_{sub}(s), h^{(2,3,4,5)}_{sub}(s), \cdots\}$.

Disjoint Subproblems

- **Question**: Will the **addition of heuristics** from subproblem (1-2-3-4) and (5-6-7-8) give an **admissible heuristic**, considering the two subproblems are not overlapped?
- **Answer**: No, since they always **share some moves**.

- **Question**: What if **not count** those shared moves?
- **Answer**: $h^{(1,2,3,4)}_{sub}(s) + h^{(5,6,7,8)}_{sub}(s) \leq c^*(s)$ ⇒ admissible.
  - Disjoint pattern database

③ from Experience

For 8-puzzle problem:

- Solve many 8-puzzles to obtain **many examples**.
- Each **example** consists of a state from the solution path and the actual cost of the solution from that point.
- These **examples** are our '**experience**' for this problem.

- **Question**: How to learn $h(s)$ from these **experience**?

Learn Heuristics from Experience

- **Question**: What are the **good experience features**?
- **Answer**: **Relevant** to predicting the states' cost to Goal, e.g.
  - $x_1(s)$: #(displaced tiles).
  - $x_2(s)$: #(pairs of adjacent tiles) that are not adjacent in Goal state.
- **Question**: How to learn h from those **relevant experience features**?
- **Answer**: (e.g.) Construct model as

$$h(s) = w_1 x_1(s) + w_2 x_2(s),$$

where w_1, w_2 are model parameters to learn from training data by a learning method such as neural networks and decision trees.