Search Problem

Cost? → ④ Cost function

① Initial State → action 1 → State 2(1) / action 2 → State 2(1) / action n → State 2(n) → Goal Test → Yes! / No!

② Set of deterministic actions

③ ⑤

Problem-Solving Agent

Initial State → Sequence of actions → Goal State.

What agent needs to do: Search ⇒ find actions

How? Build a Search tree

General_Search(problem, strategy)

initialize the search tree using initial state.

Loop do.

if no candidates return failure.

choose a leaf node for expansion according to "Strategy" ← depth-first? / breadth-first? / ...

if ~ contains a goal state, return corresponding solution

else, add the nodes to Search tree

Uniformed Search Strategy

① Breadth-first Search

"Expand All leaf nodes at each step"

Pros:

if there is a solution, the algorithm will find it

if there are multiple solutions, algorithm will return the one with Shortest length

Cons:

Time/Space Complexity too high!

To evaluate a "Search strategy"

- Completeness: can it find a solution?
- Time Complexity:
- Space Complexity
- Optimality: if multiple solutions, can return the best one?

Depth Search

generate one successor node at a time.

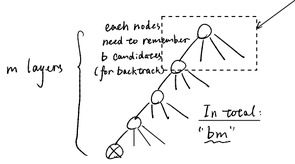
ancestor → successor → successor's successor

Depth-first vs. Breadth-search

- b: branching factor (How many "candidates" at each node).
- d: depth of solution.
- m: depth bound of depth-first search. (m > b)

	Time Complexity	Space Complexity	Optimal?	Complete?
Depth-first	b^m	bm	No.	Yes
Breadth-first	b^d	b^d	Yes	Yes.

Worse Case!



Combining Method: Iterative Deepening Search

"Successive Depth-first Search"

increasing depth bound by 1 each time.

Avoid repeated State

"A → B → A"

"Heuristic Search Strategy" 启发式搜索

Heuristic function: state → h() → "How far from a goal state"

A* search

Evaluation Function: $f(n) = g(n) + h(n)$ ("heuristic function")

"Path Cost"

- g(n): "actual cost" (path cost)
- h(n): "estimated cost"

Procedure:

1. Create Search tree T, put in Start node n_0
   Put n_0 on a list called OPEN.
2. if OPEN is empty, return failure.
3. OPEN → extract → first node n
4. if n is "goal node", return the path (solution): $n_0 \to n$.
5. else:
   Expand n → "All n's successors" → remaining nodes → add into OPEN.
   Delete those which already n's ancestor
6. Reorder (Sort) the list OPEN in ascending order of $f(n) = h(n) + g(n)$
7. back to step 2.

Remarks:

① h() is admissible: "h() ≤ actual cost"

② g() = sum of "operators along the path"

cost1 cost2 ... costn

"$g() = \sum_{i=1}^{n} cost_i$"

⇒ f(n) is monotonic, i.e. never decreasing.

③ A* is Complete:

n_0 n_1 n_2 ..... goal.

A* search → f(goal)

since f(n) is non-decreasing

④ A* is optimal:

Goal 1. Goal 2. f(n)

A* search

Will definitely find Goal 1 first

Complexity of A*:

① Can be significantly efficient than uniformed search strategy

② h(n) with higher value tend to make Search more efficient

why? Let f^* be the actual cost of goal state

☆ all nodes with f() value larger than f^* will be prune!

☆ f^* is the actual cost from n_0 to goal. which is unchanged!

∴ higher h() → more non-goal nodes be pruned

"We're using f(n) to estimate the cost!"

Since h(n) < actual cost.

f(goal) < f^*

⇒ With higher h(n), more non-goal node's f(n) value will be larger than f^*. then they will never be traversed!

⇒ h(n) ↑ efficient ↑

What we want: h(n) with estimated cost smaller than actual cost.

Meanwhile, h(n) should be close to actual cost!

h_1 h_2 h_3 h_4 Actual Cost.

Best one!

Relaxed Problem:

Actual Problem → (drop some restrictions) → Relaxed Problem

path cost: h(relaxed) < h(actual)

We can use relaxed problem to find heuristic function h(n)

But cannot be too relaxed!